Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

DeLuxing: Deep Lagrangian Underestimate Fixing for Column-Generation-Based Exact Methods

Yu Yang

To cite this article:
Yu Yang (2025) DeLuxing: Deep Lagrangian Underestimate Fixing for Column-Generation-Based Exact Methods. Operations Research 73(3):1184-1207. https://doi.org/10.1287/opre.2023.0398

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

OPERATIONS RESEARCH
Vol. 73, No. 3, May–June 2025, pp. 1184–1207
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Contextual Areas

DeLuxing: Deep Lagrangian Underestimate Fixing for Column-Generation-Based Exact Methods

Yu Yang[a]

[a] Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida 32611
Contact: yu.yang@ise.ufl.edu, https://orcid.org/0000-0002-0502-7603 (YY)

Received: July 21, 2023
Revised: May 19, 2024; August 10, 2024
Accepted: September 29, 2024
Published Online in Articles in Advance: November 27, 2024

Area of Review: Transportation

https://doi.org/10.1287/opre.2023.0398

Abstract. In this paper, we propose an innovative variable fixing strategy called *deep Lagrangian underestimate fixing* (DeLuxing). It is a highly effective approach for removing unnecessary variables in column-generation (CG)-based exact methods used to solve challenging discrete optimization problems commonly encountered in various industries, including vehicle routing problems (VRPs). DeLuxing employs a novel linear programming (LP) formulation with only a small subset of the enumerated variables, which is theoretically guaranteed to yield qualified dual solutions for computing Lagrangian underestimates (LUs). Because of their small sizes, DeLuxing can efficiently solve a sequence of similar LPs to generate multiple high-quality LUs, and thus can, in most cases, remove over 75% of the variables from the enumerated pool. We extend the fundamental concept underpinning the new formulation to contexts beyond variable fixing, namely, variable type relaxation and cutting plane addition. We demonstrate the effectiveness of the proposed method in accelerating CG-based exact methods via the capacitated multitrip vehicle routing problem with time windows (CMTVRPTW), its two important variants with loading times or release dates, and the classic capacitated VRP (CVRP) and VRPTW. Enhanced by DeLuxing and the extensions, one of the best exact methods for solving the CMTVRPTW doubles the size of instances solved optimally for the first time while being more than 7 times on average and up to over 20 times as fast as top-performing exact methods reported in the literature. It further accelerates RouteOpt, one of the world's fastest VRP solvers, by 45% and 71%, respectively, for solving the 200-node CVRP and 300-node VRPTW instances. Although DeLuxing is less effective for longer routes, it still contributes to an effective primal heuristic.

Funding: This research was supported by the National Science Foundation, Division of Civil, Mechanical and Manufacturing Innovation [Grant CMMI-2309667], and Alibaba Group US, DAMO Academy [Grant AGR00022274].

Supplemental Material: All supplemental materials, including the computer code and data that support the findings of this study, are available at https://doi.org/10.1287/opre.2023.0398.

Keywords: column generation • variable fixing • Lagrangian underestimate • multitrip vehicle routing

1. Introduction

The textbook Dantzig-Wolfe decomposition (DWD; Dantzig and Wolfe 1960) naturally gives rise to a column generation (CG) approach for solving challenging linear programs (LPs), where "promising" variables are generated and added to the restricted master program (RMP) as needed throughout the solution process. This idea of implicitly dealing with variables when there are too many of them dates back to Ford and Fulkerson (1958) and has expanded well beyond the original context of LP solving. In particular, it has been successfully combined with the well-known branch-and-bound framework (Land and Doig 2010) into the branch-and-price (BP) approach (Barnhart et al. 1998) and additionally with problem-specific cutting planes into the branch-price-and-cut (BPC) approach (e.g., Kohl et al. 1999, Fukasawa et al. 2006, Jepsen et al. 2008) for solving integer programs (IPs). BP and BPC methods are now the leading exact algorithms to approach various challenging discrete optimization problems arising in the industry, including vehicle routing problems (VRPs; Desaulniers et al. 2016a, Pessoa et al. 2020), inventory

routing problems (Engineer et al. 2012, Desaulniers et al. 2016b), and crew rostering problems (CRPs; Quesnel et al. 2020, Breugem et al. 2022).

A persistent challenge associated with CG is its tendency to generate an excessive number of columns when solving large-scale instances, which slows down the solution process and consumes a large amount of memory. To alleviate this problem, it is possible to adopt a straightforward column clean-up procedure that drops columns with large reduced costs (see section 5.2 of Pessoa et al. 2020) at the expense of more CG iterations required to solve the LPs optimally. To further mitigate the issue, most BPC methods incorporate reduced cost fixing (RCF), which fixes variables based on their reduced costs, as a default functionality to remove variables (Pecin et al. 2017a, b). RCF builds upon the fact that the reduced cost of a given variable x_i lower bounds the absolute change in the optimal value of the LP relaxation for each unit change in the value of x_i. The variable bound can thus be tightened accordingly to prevent the LP from achieving objective values worse than a known primal bound of the mixed integer program (MIP; for more details, see Wolsey and Nemhauser 1999, p. 389). For problems involving binary variables (e.g., Crowder et al. 1983, Johnson et al. 1985), RCF can directly fix variables, and those fixed to zero can be safely removed from the formulation without compromising solution optimality.

1.1. Motivation

Compared with classic compact formulations (e.g., vehicle or commodity flow-based formulations; see Baldacci et al. 2004, Cappanera and Gallo 2004), an extensive formulation, such as a set partitioning formulation or DWD reformulation, usually produces much tighter lower bounds (by default, we consider minimization problems in this paper) but needs to be solved by a CG approach because of the exponential number of variables. Such tight lower bounds make it possible to enumerate all columns with reduced costs no larger than the current integrality gap at an early stage. The enumeration implicitly applies RCF and was first recommended by Baldacci et al. (2008) for solving VRPs, which has led to remarkable acceleration (Paradiso et al. 2020, Sadykov et al. 2021, Yang 2023).

The enumeration is usually activated when the current integrality gap drops below a threshold Δ. Thus, the aggressiveness of enumeration is directly controlled by Δ, with larger values indicating higher aggressiveness. Increasing the aggressiveness within a certain range helps to close an open branch-and-bound node (BBN) faster, although too large a Δ results in the enumeration of an excessively large column pool or even failure of enumeration because of hitting limits on, for example, time, memory, or the pool size, causing deterioration in overall performance (Costa et al. 2019). For challenging instances with a reasonable Δ, it is common to have millions of columns or more enumerated.

Although RCF can also be applied after enumeration to reduce the pool size gradually (see Pessoa et al. 2020), when the route length is not long (usually no more than 15 customers per route), its effectiveness is limited for three major reasons. First, the columns in the pool are promising ones and tend to be difficult to remove because they have implicitly passed the initial screening by RCF in the enumeration phase. Second, the effectiveness of RCF relies heavily on the changes in the lower and upper bounds (UBs). Multiple rounds of cutting plane addition and branching are typically required before the pool can be shrunk to a tractable size such that the BBN can be closed by directly solving an IP with all columns left in the pool. Consequently, RCF may iterate through the entire pool many times in this process, incurring a substantial increase in computational load. Finally, RCF usually uses only one optimal dual solution from the most recent LP, which can be somewhat arbitrary within the optimal face of the corresponding polyhedron and thus results in fluctuating and mostly mediocre performance.

Unfortunately, to the best of our knowledge, not much progress has been made in addressing the said causes of ineffectiveness, which motivates this research. Specifically, we seek to unlock the full potential of variable fixing for CG-based exact methods when an enumeration procedure is employed.

1.2. Contributions, Limitations, and Outline

This paper proposes a *deep Lagrangian underestimate fixing* (DeLuxing) method widely applicable to accelerate CG-based exact methods. We summarize our contributions as follows.

- *We introduce a novel LP formulation that yields high-quality Lagrangian underestimates (LUs) and rigorously prove its validity.* The LP includes only a small subset of variables (i.e., those with reduced costs no larger than half of the current gap), allowing for a rapid search for promising dual solutions. The variable fixing induced by employing such dual solutions addresses the ineffectiveness of RCF from three perspectives: (i) it imposes much less restriction on qualified dual solutions whereas the standard RCF generally necessitates the use of optimal dual solutions; (ii) it takes effect at the current BBN by reducing the strong reliance on the quality of the lower bound, avoiding repeated branching or addition of cutting planes before achieving its significance; (iii) it proactively seeks multiple dual solutions to generate LUs of high quality that fix a large number of variables with mild computational overhead.
- *We expand the fundamental idea behind the proposed LP formulation to achieve additional acceleration beyond variable fixing.* More precisely, following this principle,

we prove that, under mild conditions, a large proportion of the integer variables can be relaxed to continuous ones in the RMP, thereby expediting the computation. Additionally, we enhance the iterative process of adding cutting planes by conducting computations on a restricted reformulation, which includes only a subset of all variables—those with reduced costs not exceeding half of the current gap. This new approach to cut addition further accelerates the computation while resulting in minimal or no deterioration in the quality of the obtained lower bound. It is worth mentioning that these extensions preserve the exactness of the entire algorithm.

- *We propose a straightforward yet effective algorithmic framework that systematically directs the exploration for promising dual solutions, and we provide insights into the mechanisms contributing to its success.* The key idea involves bundling columns with similar characteristics into a reference point to guide the search. A clustering procedure is initially employed to identify Euclidean distance-based similarities among columns. The algorithm then starts from each cluster and conducts a deep search using a reference point computed with newly identified removable columns at each iteration until a stopping criterion is reached. One can view this iterative procedure as implicitly revealing the similarities among columns via the reference points.
- *We demonstrate that DeLuxing, as a versatile variable screening tool, effectively removes unnecessary variables and can be flexibly applied throughout a BPC method.* Our experiments on the capacitated multitrip vehicle routing problem with time window (CMTVRPTW) show that DeLuxing can remove over 75% of the variables in most cases. Its effectiveness is even more pronounced as the problem size increases, achieving a reduction of up to 99%. In addition to its standard usage of removing variables after an exact enumeration, DeLuxing can serve as a crucial component in a highly effective primal heuristic.
- *We conduct an extensive numerical study and show that DeLuxing, along with several acceleration techniques inspired by it, takes the performance of BPC methods to an entirely new level.* One of the best exact methods for solving the CMTVRPTW in Yang (2023) enhanced by the proposed DeLuxing can solve all instances with 140 customers for the first time, doubling the size of instances that can be solved to optimality. Furthermore, it achieves near-optimal solutions with an average optimality gap of 0.5% for instances with up to 200 customers. Additionally, remarkable acceleration (45% and 71%, respectively) is achieved on top of RouteOpt when solving the 200-node capacitated VRP (CVRP) and 300-node VRPTW instances.

The effectiveness of DeLuxing relies on enumerating all necessary columns early in the optimization process, as seen in the CMTVRPTW and its two variants. For classic VRPs (e.g., CVRP and VRPTW) with short to moderate routes, considerable computation is still required after enumeration, and DeLuxing can significantly accelerate the solution process. However, in instances where the route length is excessively long, enumeration may not succeed until a very late stage. A node can be closed rapidly without much branching after a successful enumeration. In such cases, DeLuxing may not be particularly effective. Nonetheless, the primal heuristic inspired by DeLuxing demonstrates remarkable effectiveness even for long-route instances.

The rest of the paper is structured as follows. Section 2 provides a review of some variable fixing techniques related to RCF. Section 3 describes preliminaries on the enumeration procedure and variable fixing techniques. Section 4 introduces the theoretical foundations and relevant formulations for dual picking, and gives an overview of DeLuxing. A detailed explanation of DeLuxing is provided in Section 5. Three extensions inspired by DeLuxing are presented in Section 6. We report the results of five sets of numerical experiments in Section 7. Finally, in Section 8, we make concluding remarks and identify potential avenues for future research. The detailed numerical results, the source code, and the data needed to reproduce these results can be found in the e-companion.

2. Literature Review

In this section, we review variable fixing techniques that rely on the well-known RCF, specifically focusing on those integrated into customized CG-based algorithms. Additionally, we discuss some recent efforts to enhance the effectiveness of RCF in more general settings. Finally, we review the current state-of-the-art exact methods for solving the CMTVRPTW, its two variants, CVRP, and VRPTW to facilitate the comprehension of our numerical results in Section 7.

The idea of what is now known as RCF was originally introduced in the seminal work of Dantzig et al. (1954) for solving the traveling salesman problem. Its practical effectiveness and ease of implementation have made it a standard procedure in cutting-edge MIP solvers such as Gurobi (Achterberg 2018), CPLEX (Bixby et al. 2000), and SCIP (Achterberg et al. 2008). Moreover, RCF has been applied to leverage the strengths of MIP in a constraint programming (CP) framework (Yunes et al. 2010, Bacchus et al. 2017). Beyond MIP and CP, RCF has also been employed in two-stage stochastic programming (Crainic et al. 2018), semidefinite relaxation (Posta et al. 2012), and many others.

However, applying RCF to fix nominal variables in an extensive formulation solved by a CG-based method cannot be done blindly as it necessitates restructuring the pricing subproblem to prevent the regeneration of eliminated variables. Therefore, fixing by reduced costs

is typically applied to implicit variables, which is equivalent to removing a subset of the variables in the RMP. Irnich et al. (2010) propose to use path-reduced cost to remove arcs from the underlying network of routing and scheduling problems without sacrificing optimality. They derive analytical results on which dual solutions *to the pricing subproblem* can achieve the largest reduced cost for an arc-flow variable under a given dual solution to the LP relaxation of the RMP. The authors conclude that approximately 80% of the arcs can be eliminated when the optimality gap is around 1%. This arc elimination technique has also been successfully applied in Pecin et al. (2017a) for solving the VRPTW.

Pessoa et al. (2010) and Pecin et al. (2017b) refine this approach to a resource-value-dependent arc elimination procedure for solving parallel machine scheduling problems and the CVRP, respectively. Using a similar approach, Sadykov et al. (2021) perform the so-called bucket arc elimination on a sophisticated way of organizing labels in the labeling algorithm called bucket graph. They report a 6% speedup compared with a standard arc elimination procedure independent of resources and conclude that hard instances with small primal-dual gaps benefit more from this new bucket arc elimination. Desaulniers et al. (2020) propose to generalize the idea to fix sequences of two arcs with a modification in the labeling algorithm for pricing. Experiments on the VRPTW and four variants of the electric VRPTW show that single-arc fixing can eliminate more than 90% of the feasible two-arc sequences, and two-arc sequence fixing can fix approximately half of the remaining ones, achieving an overall reduction of around 19% in the BPC computation time.

Enumeration, which identifies all potential columns with reduced costs not exceeding the current integrality gap, is another effective way to utilize RCF. This procedure has been employed in many high-performing BPC approaches (e.g., Baldacci et al. 2011a, b, c, d, 2013; Pessoa et al. 2020; Yang 2023) since its inception in Baldacci et al. (2008). After enumeration, RCF can be applied to the nominal variables in the conventional manner as columns are no longer generated by the labeling algorithm. Nonetheless, the efficacy of RCF is limited, especially at the current BBN, because of the three reasons explained in Section 1.1, which leaves room for improving RCF when applied in this way. It has been observed in Sellmann (2004) that distinct dual solutions can result in significantly different effectiveness and suboptimal dual solutions could potentially result in even more variable fixing than optimal ones, which suggests a promising research avenue.

Bajgiran et al. (2017) take a step in exploring such improvement and propose to search for a dual solution maximizing the number of variables that can be fixed by solving an MIP. Their experiments demonstrate that the new dual picking method yields an average speedup of 20% in geometric mean over the default CPLEX. The authors also observe that by limiting the search to the optimal dual face instead of the entire dual feasible space, almost the same amount of fixing can be achieved while being orders of magnitude faster. However, solving the MIP constructed by Bajgiran et al. (2017) can be challenging as it includes n binary variables, where n is the number of variables in the original problem. The authors thus set a time limit of 10 minutes and use all feasible solutions obtained in the process for variable fixing. In Yang (2023), the author proposes to solve a new auxiliary problem that computes a second dual solution for fixing variables after enumeration in the price-cut-and-enumerate method (EPCEM) for the CMTVRPTW. Based on a similar idea, de Lima et al. (2023) develop two strategies to compute alternative dual solutions for variable fixing when dealing with network flow models. It is worth mentioning that because they consider the DWD reformulation, the variable fixing is conducted on the arcs of the underlying network. These methods can be performed iteratively, with each iteration building upon the previous round of variable fixing. They demonstrate that these techniques speed up the proof of optimality despite their high computational overhead. Lastly, Mingozzi et al. (2013) propose four bounding procedures to enhance lower bounds in solving multitrip VRPs, where the dual solution from each procedure is additionally utilized for RCF. Note that they do not actively search for new dual solutions for variable fixing purposes.

Our proposed DeLuxing method eliminates nominal variables in the RMP via multiple dual solutions. It fundamentally differs from existing approaches in several key aspects. First, DeLuxing uses a novel small-sized LP formulation to search for qualified dual solutions that are not restricted to be (sub-)optimal or feasible for the original dual problem. Second, it uses a completely new way of searching that exploits the underlying column similarities revealed iteratively. Last, DeLuxing does not rely on specific problem structures, unlike previous approaches such as those in de Lima et al. (2023), making it generally applicable to both CG-based exact methods and MIPs.

The CMTVRPTW is an increasingly popular extension of the well-known capacitated VRPTW by allowing each vehicle to perform multiple trips, catering to real-world scenarios with constraints such as limited vehicles, driver availability, vehicle capacity, or trip duration (Cattaruzza et al. 2016b). In this study, we also consider two notable variants of the CMTVRPTW: the CMTVRPTW *with loading times* (CMTVRPTW-LT; Hernandez et al. 2016) and the CMTVRPTW *with release dates* (CMTVRPTW-R; Cattaruzza et al. 2016a). In the CMTVRPTW-LT, a loading time, computed as a fixed percentage of the total service time of all customers visited in the route, is required at the depot before a vehicle

can start the trip. In the CMTVRPTW-R, the goods requested by each customer become available at the depot at possibly different times, called release dates. Consequently, a vehicle cannot depart from the depot until all the goods to be aboard are available, that is, its departure time should be no earlier than the latest release date among the goods it ships. Although the loading times and release dates do not affect the objective, which is to minimize the total distance traveled by all vehicles, they impact the solution space and thus the problem difficulty.

Paradiso et al. (2020) develop the first exact solution framework (ESF) capable of solving the CMTVRPTW and its four variants including the CMTVRPTW-LT and CMTVRPTW-R. The ESF, operating as a BPC type of algorithm, solves a novel structure-based formulation, which significantly outperforms all previously leading exact methods such as those proposed by Hernandez et al. (2014, 2016) and Cheng et al. (2020). Inspired by Paradiso et al. (2020), Yang (2023) introduces a superstructure-based formulation with fewer variables while maintaining the same theoretical tightness. The EPCEM developed therein demonstrates substantially superior numerical performance compared with the ESF. Two very recent studies achieve remarkable advancements, representing the state of the art. Roboredo et al. (2023) propose a new graph-based model, effectively solved by the VRP-Solver developed in Pessoa et al. (2020) with a simple parameterization. Zhang (2022) proposes a novel three-phase exact method for solving all five problems. In the initial two phases, a route-based and a trip-based model are solved, respectively, to obtain tight lower bounds, followed by solving a trip-based model with dynamic time discretization in the last phase. In contrast, our approach works only with route-based models. Specifically, incorporating DeLuxing and the three extensions developed in this paper into the EPCEM (see Section 7.2), we obtain an enhanced exact method, denoted as Default, expanding the frontiers of this field even further. For the classic CVRP and VRPTW, the VRP-Solver (Pessoa et al. 2020) consistently achieves one of the best results in the literature and thus serves as a benchmark for experiments in Section 7.5.

3. Preliminaries

In this section, we first formally describe the enumeration procedure that is now widely applied in the BPC framework for solving challenging discrete optimization problems. Then, we review the general variable fixing technique by Lagrangian bounds. Lastly, we discuss a special variable fixing strategy via dual picking introduced in Yang (2023) and its major drawbacks, which serve as a natural motivation for this study. Throughout the paper, we use $\mathbb{R}_+$, $\mathbb{Z}$, and $\mathbb{Z}_+$ to denote the set of nonnegative real numbers, integers, and nonnegative integers, respectively. Lowercase letters in bold are used to represent vectors. The inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is denoted by $\langle \mathbf{x}, \mathbf{y} \rangle$.

3.1. The Enumeration Procedure

Consider the following extensive formulation that can be a standard set partitioning formulation or a DWD reformulation. Because the proposed method will be applied exclusively to fix integer variables, we omit continuous variables in the presentation without loss of generality. Moreover, we only consider that all variables are nonnegative in our presentation. This requirement is not necessary and can be removed through a straightforward variable substitution.

$$\begin{aligned} \mathrm{F}(\mathcal{R}): \qquad z^* = \min \quad & \sum_{r \in \mathcal{R}} c_r x_r \\ \text{s.t.} \quad & \sum_{r \in \mathcal{R}} a_{ir} x_r = b_i, \ \forall i \in \mathcal{N}, \\ & x_r \in \mathbb{Z}_+, \qquad \forall r \in \mathcal{R}. \end{aligned}$$

In the context of VRPs, $\mathcal{N}$ represents the set of customers, each of which should be visited exactly once (i.e., $b_i = 1$ for $i \in \mathcal{N}$), $\mathcal{R}$ consists of all feasible routes and possibly some relaxed routes that are not necessarily feasible (e.g., *ng*-routes from Baldacci et al. 2012), x_r is a binary decision variable taking the value of one if route $r \in \mathcal{R}$ is used and zero otherwise, and c_r and a_{ir} denote the cost and the number of times customer i is visited by route r, respectively. Because of the exponential size of $\mathcal{R}$, formulation $\mathrm{F}(\mathcal{R})$ is typically solved by a BPC method (Fukasawa et al. 2006, Pecin et al. 2017b).

Let lb and ub, separately, be a lower bound and an upper bound of the optimal value z^*. An lb is usually obtained by solving some LP relaxations of $\mathrm{F}(\mathcal{R})$, and a ub is usually set to the objective value of the best feasible solution found so far. The optimality gap, denoted by g, is computed as the difference between ub and lb, that is, $g := ub - lb$. A BPC method can try to enumerate all variables with reduced costs no larger than g with respect to (w.r.t.) the current dual solution when the gap g falls below some prespecified threshold. This idea was first proposed for solving VRPs in Baldacci et al. (2008) and has been successfully applied in most state-of-the-art BPC methods.

A labeling algorithm, slightly modified from the one used for generating columns in the pricing procedure, is typically employed to enumerate all qualified columns. Specifically, for VRPs, the dominance rule must be changed to allow dominance only between partial routes (labels) that have visited exactly the same subset of customers, ensuring that no qualified routes are missed. Additionally, the focus should be solely on elementary

routes, as relaxed routes (e.g., *ng*-routes) that visit some customers more than once are not feasible. More details can be found in Baldacci et al. (2008) and Pecin et al. (2017b). However, because of this weaker dominance rule and the requirement for elementariness, the enumeration process generally requires significantly more time than the pricing procedure. Arc elimination based on RCF (Irnich et al. 2010, Pessoa et al. 2020) can substantially reduce the network size, and therefore, performing enumeration on this reduced network can significantly shorten the computational time. This arc elimination has been implemented in RouteOpt for the CVRP and VRPTW. However, we did not apply it in our code for solving the CMTVRPTW and its two variants because enumeration only takes up a noticeable portion of the total time in easy instances that can be solved optimally within five minutes. We believe arc elimination has the potential to further accelerate the solution process for these instances and will be considered in future studies.

Let $\underline{\mathcal{R}}$ denote the set of variables that have been enumerated. In the case of VRPs, the set $\underline{\mathcal{R}}$ only consists of qualified elementary routes. RCF guarantees that solving $\mathrm{F}(\underline{\mathcal{R}})$ will yield an optimal solution to $\mathrm{F}(\mathcal{R})$ because a variable with a reduced cost greater than g cannot take a positive integer value in any optimal solution. When the cardinality of $\underline{\mathcal{R}}$ is in the tens of thousands, solving $\mathrm{F}(\underline{\mathcal{R}})$ as an IP by a general MIP solver such as Gurobi (Gurobi Optimization, LLC 2023) can yield an optimal solution within a reasonable time frame. In the case of $|\underline{\mathcal{R}}|$ being too large, the algorithm can continue the BPC procedure using inspection for CG (Contardo and Martinelli 2014). Specifically, instead of running the dynamic programming-based labeling algorithm, which can be computationally intensive, especially when many nonrobust cuts (Pecin et al. 2017b) have been added, pricing is done by evaluating the reduced costs of the columns in the pool. In both cases, the pool size significantly impacts the time required to prove optimality.

3.2. Variable Fixing by Lagrangian Bounds

A natural way to reduce the computational burden after enumeration is to remove variables from $\mathrm{F}(\underline{\mathcal{R}})$. Consider the following LP relaxation of $\mathrm{F}(\underline{\mathcal{R}})$ with cutting planes added in the solution process.

$$
\begin{aligned}
\overline{\mathrm{F}}(\underline{\mathcal{R}}): \quad \overline{z}^* = \min \quad & \sum_{r\in\underline{\mathcal{R}}} c_r x_r \\
\text{s.t.} \quad & \sum_{r\in\underline{\mathcal{R}}} a_{ir} x_r = b_i, && \forall i \in \mathcal{N}, \quad (1)\\
& \sum_{r\in\underline{\mathcal{R}}} a_{kr} x_r \le b_k, && \forall k \in \mathcal{K}, \quad (2)\\
& x_r \ge 0, && \forall r \in \underline{\mathcal{R}},
\end{aligned}
$$

where $\mathcal{K}$ denotes the index set of the added cuts. For VRPs, they typically include the rounded capacity cuts (RCCs; Laporte and Nobert 1983, Lysgaard et al. 2004), the (limited memory) subset row cuts (SRCs; Jepsen et al. 2008, Pecin et al. 2017b), and problem-specific feasibility cuts, for example, the relaxed (super-)structure feasibility cuts for the CMTVRPTW (Paradiso et al. 2020, Yang 2023).

Fixing variables by Lagrangian bounds is a widely applied technique in solving discrete optimization problems (e.g., Balas and Saltzman 1991, Balas and Carrera 1996, Holmberg and Yuan 2000). The general idea is that when a variable is set to a given value, if the Lagrangian bound is larger than the best upper bound, then this value can be excluded from the variable's feasible region. More precisely, consider the following Lagrangian dual function obtained from $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ by dualizing the constraints.

$$
\mathcal{L}(\mathbf{y},\mathbf{x}) = \sum_{i\in\mathcal{I}} b_i y_i + \sum_{r\in\underline{\mathcal{R}}} \left(c_r - \sum_{i\in\mathcal{I}} a_{ir} y_i \right) x_r,
$$

where $\mathcal{I} := \mathcal{N} \cup \mathcal{K}$, y_i for $i \in \mathcal{I}$ are the dual variables associated with Constraints (1) and (2), $\mathbf{y} = (y_i)_{i\in\mathcal{I}}$, and $\mathbf{x} = (x_r)_{r\in\underline{\mathcal{R}}}$. For convenience, we define the set $\mathcal{Y} := \{\mathbf{y} \in \mathbb{R}^{|\mathcal{I}|} : y_k \le 0, \ \forall k \in \mathcal{K}\}$. Let $\overline{\mathrm{F}}(\underline{\mathcal{R}})|_{x_j=v}$ be the formulation obtained by adding an additional constraint $x_j = v$ to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$, and $\overline{z}^*|_{x_j=v}$ be its optimal value, which is set to $+\infty$ in case of infeasibility. Because of LP weak duality, it follows that $\max_{\mathbf{y}\in\mathcal{Y}} \min_{\mathbf{x}\ge 0, x_j=v} \mathcal{L}(\mathbf{y},\mathbf{x}) \le \overline{z}^*|_{x_j=v}$. If, for any given dual vector $\hat{\mathbf{y}} \in \mathcal{Y}$, we have $\min_{\mathbf{x}\ge 0, x_j=v} \mathcal{L}(\hat{\mathbf{y}},\mathbf{x}) > ub$, then it immediately leads to $\overline{z}^*|_{x_j=v} \ge \max_{\mathbf{y}\in\mathcal{Y}} \min_{\mathbf{x}\ge 0, x_j=v} \mathcal{L}(\mathbf{y},\mathbf{x}) \ge \min_{\mathbf{x}\ge 0, x_j=v} \mathcal{L}(\hat{\mathbf{y}},\mathbf{x}) > ub$. Therefore, x_j cannot equal v in any optimal solution to $\mathrm{F}(\underline{\mathcal{R}})$. RCF can be viewed as a special case of this general technique. Specifically, if $\min_{\mathbf{x}\ge 0, x_j=1} \mathcal{L}(\mathbf{y}^*,\mathbf{x}) > ub$ for an optimal dual solution $\mathbf{y}^*$, then the binary variable x_j can be fixed to zero and thus removed.

3.3. Variable Fixing by Dual Picking

The Lagrangian bound $\min_{\mathbf{x}\ge 0, x_j=1} \mathcal{L}(\mathbf{y},\mathbf{x})$ depends on the dual $\mathbf{y}$ used, and it reduces to $\overline{z}^* + \overline{c}_j + \min_{\mathbf{x}\ge 0} \sum_{r\in\underline{\mathcal{R}}} \overline{c}_r x_r$ when $\mathbf{y}$ is an optimal dual solution to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$, where $\overline{c}_r = c_r - \sum_{i\in\mathcal{I}} a_{ir} y_i = c_r - \langle \mathbf{y}, \mathbf{a}_r \rangle$ is the corresponding reduced cost of x_r, and $\mathbf{a}_r = (a_{ir})_{i\in\mathcal{I}}$. Let $\mathcal{Y}^*$ be the set of optimal dual solutions to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$. In Yang (2023), the author proposes to pick a special point in $\mathcal{Y}^*$ to obtain large reduced costs, thereby fixing a large number of columns to zero. This involves solving the following LP, denoted by DF, that maximizes the sum of the reduced costs of all variables. According to the LP duality theorem, it is equivalent to solving AUX in the primal space (see section 6.5 of Yang

2023 for details).

(DF) :

$$\begin{cases} \max & \sum_{r\in\underline{\mathcal{R}}}\left(c_r-\sum_{i\in\mathcal{I}}a_{ir}y_i\right) \\ \text{s.t.} & \sum_{i\in\mathcal{I}}a_{ir}y_i \le c_r, \quad \forall r\in\underline{\mathcal{R}}, \\ & \sum_{i\in\mathcal{I}}b_i y_i=\overline{z}^*, \\ & y_i\le 0, \quad \forall i\in\mathcal{K}. \end{cases}$$

(AUX) :

$$\begin{cases} \min & \sum_{r\in\underline{\mathcal{R}}}c_r(x_r+1)+\overline{z}^* w \\ \text{s.t.} & \sum_{r\in\underline{\mathcal{R}}}a_{ir}x_r+b_i w=-\sum_{r\in\underline{\mathcal{R}}}a_{ir}, & \forall i\in\mathcal{N}, \\ & \sum_{r\in\underline{\mathcal{R}}}a_{kr}x_r+b_k w\le -\sum_{r\in\underline{\mathcal{R}}}a_{kr}, & \forall k\in\mathcal{K}, \\ & x_r\ge 0, & \forall r\in\underline{\mathcal{R}}. \end{cases}$$

3.3.1. Major Drawbacks. The above approach has several drawbacks. First, by design, AUX searches within the dual optimal face, using a dual solution from $\mathcal{Y}^*$ to update the reduced costs. However, $\mathcal{Y}^*$ only constitutes a small portion of all feasible dual solutions to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$, so the number of variables that can be removed by solving AUX may be limited. Second, solving AUX, possibly with different objective coefficients, to obtain multiple dual solutions can lead to more variable fixings. However, AUX has $(|\underline{\mathcal{R}}|+1)$ variables, which can easily top tens of millions for challenging instances, making AUX time-consuming and memory-intensive to solve, particularly for interior point methods that are known to outperform the simplex method for large-sized LPs. Consequently, it is computationally prohibitive to repeatedly solve AUX with varied objective coefficients.

In fact, for each individual $r\in\underline{\mathcal{R}}$, we want to maximize the reduced cost $\overline{c}_r$, which can be achieved by solving an AUX with $\overline{c}_r$ as the objective function. Thus, it requires solving AUX by a total of $|\underline{\mathcal{R}}|$ times and is computationally intractable. Instead, maximizing the sum $\sum_{r\in\underline{\mathcal{R}}}\overline{c}_r$ can be viewed as a coarse approximation that works reasonably well when the set of $|\mathcal{I}|$-dimensional vectors, $\{-\mathbf{a}_r : r\in\underline{\mathcal{R}}\}$, have some nice structure. For example, when they are close to the ray generated by $\mathbf{y}$ as depicted in the left panel of Figure 1, where $\mathbf{y}$ is an extreme point of the polyhedron $\mathcal{Y}^*$, almost all reduced costs $\overline{c}_r$ are maximized at the same extreme point $\mathbf{y}$.

However, this approach can be problematic, particularly when $-\mathbf{a}_r$ for $r\in\underline{\mathcal{R}}$ are scattered in the $|\mathcal{I}|$-dimensional Euclidean space. Let $\mathbf{o}' := \frac{1}{|\underline{\mathcal{R}}|}\sum_{r\in\underline{\mathcal{R}}}(-\mathbf{a}_r)$ be the center and $r' := \max_{r\in\underline{\mathcal{R}}}\|-\mathbf{a}_r-\mathbf{o}'\|$ be the radius. Maximizing the sum $\sum_{r\in\underline{\mathcal{R}}}\overline{c}_r$ essentially maximizes the inner product $\langle\mathbf{y},\mathbf{o}'\rangle$ for $\mathbf{y}\in\mathcal{Y}^*$, which is achieved at extreme point $\mathbf{y}^1$. However, as shown in the right panel of Figure 1, when the radius r' is relatively large, $\mathbf{y}^1$ may not be the maximizer for a majority of $\overline{c}_r$. For instance, all the purple (triangle) and yellow (square) points are maximized at extreme points $\mathbf{y}^2$ and $\mathbf{y}^3$, respectively. As a result, some variables could have been fixed if a better dual solution, such as $\mathbf{y}^2$ or $\mathbf{y}^3$ in this example, had been used to compute the reduced costs.

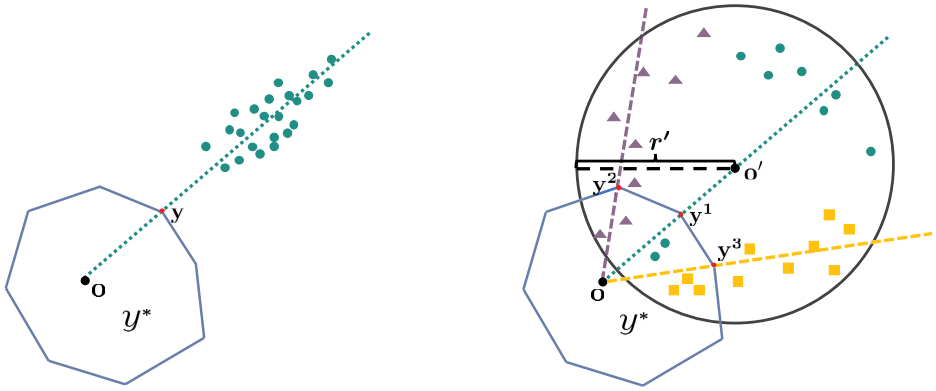
4. The DeLuxing

The proposed DeLuxing aims to overcome the previously mentioned limitations. More specifically, DeLuxing enables the removal of variables by using LUs computed with dual solutions that are not necessarily optimal and may even be infeasible, which enlarges the search space substantially. In this process, a sequence of carefully crafted LPs of much smaller size than the AUX is solved instead of just solving a single AUX, significantly increasing the chance of an unnecessary variable being removed. According to our numerical experiments detailed in Section 7, DeLuxing can remove more than 75% of the columns in most cases, reducing $\underline{\mathcal{R}}$ to a quarter or less of its original size.

4.1. Theoretical Foundations

Constructing small-sized LPs to obtain multiple dual solutions fast is one of the key ideas behind DeLuxing, which is motivated by the observation that the number of variables with reduced costs no greater than $\frac{g}{2}$ only comprises a small proportion (mostly less than 15%) of the elements in $\underline{\mathcal{R}}$. This ratio is observed to be even smaller for larger instances. In other words, $|\underline{\mathcal{R}}_2^{\le}|$ is much smaller than $|\underline{\mathcal{R}}|$, where $\underline{\mathcal{R}}_2^{\le} := \{r\in\underline{\mathcal{R}} : \overline{c}_r^{\boldsymbol{\pi}} \le \frac{g}{2}\}$, $\boldsymbol{\pi}$ is a given optimal dual solution to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$, and $\overline{c}_r^{\boldsymbol{\pi}} = c_r - \langle\boldsymbol{\pi},\mathbf{a}_r\rangle$ is the reduced cost of variable x_r w.r.t. $\boldsymbol{\pi}$. Although we use the simplified notation $\underline{\mathcal{R}}_2^{\le}$ that does not explicitly show its dependence on $\boldsymbol{\pi}$, it should not cause any confusion. Once $\underline{\mathcal{R}}_2^{\le}$ is computed using a given $\boldsymbol{\pi}$, it will not be recomputed w.r.t. another dual solution unless otherwise specified. Thus, it is expected that substantial acceleration will be achieved if the computation can be performed using solely variables x_r for r in set $\underline{\mathcal{R}}_2^{\le}$ instead of the whole set $\underline{\mathcal{R}}$. This is made possible by the following Lemma 1 (proposition 4 from Yang 2023), which ensures that any optimal solution to $\mathrm{F}(\underline{\mathcal{R}})$ can have $(k-1)$ variables with reduced costs larger than $\frac{g}{k}$ taking positive integer values.

Figure 1. (Color online) Example Illustrating that a Direct Maximization of the Sum of All Reduced Costs Can Be Problematic



Note. **o** represents the origin; $\mathcal{Y}^*$ represents the feasible region of DF; $\mathbf{y}$, $\mathbf{y}^1$, $\mathbf{y}^2$, and $\mathbf{y}^3$ are extreme points of $\mathcal{Y}^*$; and the points in green (circle), purple (triangle), and yellow (square) represent $-\mathbf{a}_r$ for some $r \in \underline{\mathcal{R}}$.

Lemma 1 (Proposition 4 in Yang 2023). *For any given positive integer k, the inequality* $\sum_{r\in\underline{\mathcal{R}}_k^>} x_r \le k-1$ *is valid for* $F(\underline{\mathcal{R}})$, *where* $\underline{\mathcal{R}}_k^> := \{r \in \underline{\mathcal{R}} : \bar{c}_r^{\boldsymbol{\pi}} > \frac{g}{k}\}$.

Evidently, Lemma 1 also reduces to the standard RCF when $k = 1$. We consider the case when $k = 2$, and work with the formulation $\overline{\mathrm{F}}(\underline{\mathcal{R}}_2^{\le})$ obtained from $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ with the set of variables x_r for $r \in \underline{\mathcal{R}}$ replaced by $r \in \underline{\mathcal{R}}_2^{\le}$. Let $\mathcal{Y}^{\boldsymbol{\pi}}$ be the set of all feasible dual solutions to $\overline{\mathrm{F}}(\underline{\mathcal{R}}_2^{\le})$, that is, $\mathcal{Y}^{\boldsymbol{\pi}} := \{\mathbf{y} \in \mathbb{R}^{|\mathcal{I}|} : \sum_{i\in\mathcal{I}} a_{ir} y_i \le c_r,\ \forall r \in \underline{\mathcal{R}}_2^{\le},\ y_i \le 0,\ \forall i \in \mathcal{K}\}$.

Proposition 1. *For any given* $\hat{\mathbf{y}} \in \mathcal{Y}^{\boldsymbol{\pi}}$ *and* $j \in \underline{\mathcal{R}}_2^>$, *if* $\langle \mathbf{f}^j, \hat{\mathbf{y}}\rangle > ub - c_j$ *is satisfied, where* $\mathbf{f}^j = (b_i - a_{ij})_{i\in\mathcal{I}}$, *then variable* x_j *can be removed from formulation* $\mathrm{F}(\underline{\mathcal{R}})$.

Proof. Let $\mathrm{F}'(\underline{\mathcal{R}})$ be the formulation obtained from $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ by adding the additional constraint $\sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1$. Because of Lemma 1, we know $\sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1$ is valid for $\mathrm{F}(\underline{\mathcal{R}})$. Therefore, $\mathrm{F}'(\underline{\mathcal{R}})$ can be viewed as a relaxation of $\mathrm{F}(\underline{\mathcal{R}})$. Let $z'|_{x_j=1}$ and $z^*|_{x_j=1}$ be the optimal value of $\mathrm{F}'(\underline{\mathcal{R}})$ and $\mathrm{F}(\underline{\mathcal{R}})$, respectively, when $x_j = 1$ is enforced for a given $j \in \underline{\mathcal{R}}_2^>$. Then we have $z'|_{x_j=1} \le z^*|_{x_j=1}$. For convenience, we define $\bar{c}_r^{\hat{\mathbf{y}}} := c_r - \langle \hat{\mathbf{y}}, \mathbf{a}_r\rangle$. Note that $\underline{\mathcal{R}}_2^{\le} = \underline{\mathcal{R}} \setminus \underline{\mathcal{R}}_2^>$.

$$\begin{aligned}
&\min_{\substack{\mathbf{x}\ge 0, x_j=1,\\ \sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{x}) = \min_{\substack{\mathbf{x}\ge 0, x_j=1,\\ \sum_{r\in\underline{\mathcal{R}}_2^>\setminus\{j\}} x_r = 0}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{x}) \\
&= \sum_{i\in\mathcal{I}} b_i \hat{y}_i + \min_{\substack{\mathbf{x}\ge 0, x_j=1,\\ \sum_{r\in\underline{\mathcal{R}}_2^>\setminus\{j\}} x_r = 0}} \sum_{r\in\underline{\mathcal{R}}} \left(c_r - \sum_{i\in\mathcal{I}} a_{ir}\hat{y}_i\right) x_r \\
&= \sum_{i\in\mathcal{I}} b_i \hat{y}_i + \bar{c}_j^{\hat{\mathbf{y}}} + \min_{x_r\ge 0, \forall r\in\underline{\mathcal{R}}_2^{\le}} \sum_{r\in\underline{\mathcal{R}}_2^{\le}} \bar{c}_r^{\hat{\mathbf{y}}} x_r \\
&\ge \sum_{i\in\mathcal{I}} b_i \hat{y}_i + \bar{c}_j^{\hat{\mathbf{y}}} = c_j + \sum_{i\in\mathcal{I}} (b_i - a_{ij})\hat{y}_i \\
&= c_j + \langle \mathbf{f}^j, \hat{\mathbf{y}}\rangle > ub
\end{aligned}$$

where the inequality is because of $\bar{c}_r^{\hat{\mathbf{y}}} \ge 0$ for $r \in \underline{\mathcal{R}}_2^{\le}$, given $\hat{\mathbf{y}}$ is a feasible dual to $\overline{\mathrm{F}}(\underline{\mathcal{R}}_2^{\le})$. Consequently,

$$\begin{aligned}
z^*|_{x_j=1} \ge z'|_{x_j=1} &= \bar{z}^*|_{x_j=1,\ \sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1} \\
&\ge \max_{\mathbf{y}\in\mathcal{Y}} \min_{\substack{\mathbf{x}\ge 0, x_j=1,\\ \sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1}} \mathcal{L}(\mathbf{y}, \mathbf{x}) \\
&\ge \min_{\substack{\mathbf{x}\ge 0, x_j=1,\\ \sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{x}) > ub,
\end{aligned}$$

where $\bar{z}^*|_{x_j=1,\ \sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1}$ is the optimal value of $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ when $x_j = 1$ and $\sum_{r\in\underline{\mathcal{R}}_2^>} x_r \le 1$ are enforced, and the second inequality is because of weak duality. We conclude that any feasible solution to $\mathrm{F}(\underline{\mathcal{R}})$ with x_j equal to one must have an objective value larger than ub, and thus x_j can be removed from the formulation, which completes the proof. □

Remark 1. According to Proposition 1, any $\hat{\mathbf{y}} \in \mathcal{Y}^{\boldsymbol{\pi}}$ can be used for removing unnecessary variables, even if it may be infeasible to the dual of $\overline{\mathrm{F}}(\underline{\mathcal{R}})$. It provides an easily verifiable criterion to decide if a variable x_j for $j \in \underline{\mathcal{R}}_2^>$ can be removed for a given $\hat{\mathbf{y}}$. Note that vectors $\mathbf{f}^j$ and values $ub - c_j$ for all $j \in \underline{\mathcal{R}}_2^>$ need to be calculated only once and can be reused throughout the computation. Upon obtaining a new feasible dual solution $\hat{\mathbf{y}}$ to $\overline{\mathrm{F}}(\underline{\mathcal{R}}_2^{\le})$, it suffices to compute the inner product $\langle \mathbf{f}^j, \hat{\mathbf{y}}\rangle$ and make the comparison. Two distinct variables, $x_i, x_j \in \underline{\mathcal{R}}$, are said to be compatible if there exists a feasible solution to $\mathrm{F}(\underline{\mathcal{R}})$ with both x_i and x_j equal to one. In other words, x_i and x_j are incompatible if $x_i + x_j \le 1$ is valid for $\mathrm{F}(\underline{\mathcal{R}})$. The following Proposition 2 provides a sufficient condition for removing a variable x_j for $j \in \underline{\mathcal{R}}_2^{\le}$.

Proposition 2. *For any given* $\hat{\mathbf{y}} \in \mathcal{Y}^{\boldsymbol{\pi}}$ *and* $j \in \underline{\mathcal{R}}_2^{\le}$, *if* $\langle \mathbf{f}^j, \hat{\mathbf{y}}\rangle > ub - c_j - \min\{\eta_j, 0\}$ *is satisfied, where* $\eta_j = \min_{r\in\mathcal{S}^j}\{c_r -$

$\langle \widehat{\mathbf{y}}, \mathbf{a}_r \rangle\}$ *and* $\mathcal{S}^j = \{r \in \underline{\mathcal{R}}_2^> : x_r$ *is compatible with* $x_j\}$, *then variable* x_j *can be removed from formulation* $\mathrm{F}(\underline{\mathcal{R}})$.

Proof. We use the notation defined in the above proof of Proposition 1. Note that $j \in \underline{\mathcal{R}}_2^{\leq}$ in this case. Additionally, let $\overline{\mathcal{S}^j} = \underline{\mathcal{R}}_2^> \setminus \mathcal{S}^j$. By the definition of $\mathcal{S}^j$, it follows that for any $j' \in \overline{\mathcal{S}^j}$, $x_j + x_{j'} \leq 1$ is valid for $\mathrm{F}(\underline{\mathcal{R}})$. Let $\mathrm{F}''(\underline{\mathcal{R}})$ be the formulation obtained from $\mathrm{F}'(\underline{\mathcal{R}})$ by adding the additional constraints $x_j + x_{j'} \leq 1$, $\forall j' \in \overline{\mathcal{S}^j}$. Let $\mathcal{X} := \{\mathbf{x} \in \mathbb{R}_+^{|\underline{\mathcal{R}}|} : \sum_{r \in \underline{\mathcal{R}}_2^>} x_r \leq 1,\ x_j = 1, x_j + x_{j'} \leq 1,\ \forall j' \in \overline{\mathcal{S}^j}\} = \{\mathbf{x} \in \mathbb{R}_+^{|\underline{\mathcal{R}}|} : \sum_{r \in \underline{\mathcal{R}}_2^>} x_r \leq 1, x_j = 1, x_{j'} = 0,\ \forall j' \in \overline{\mathcal{S}^j}\}$, and $\mathcal{X}' := \mathcal{X} \cap \{\mathbf{x} \in \mathbb{R}_+^{|\underline{\mathcal{R}}|} | : x_r = 0,\ \forall r \in \underline{\mathcal{R}}_2^{\leq}, r \neq j\}$. Let $z''|_{x_j=1}$ be the optimal value of $\mathrm{F}''(\underline{\mathcal{R}})$ when $x_j = 1$ is enforced. Because $\mathrm{F}''(\underline{\mathcal{R}})$ again can be viewed as a relaxation of $\mathrm{F}(\underline{\mathcal{R}})$, we have $z''|_{x_j=1} \leq z^*|_{x_j=1}$.

$$\begin{aligned}
\min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\widehat{\mathbf{y}}, \mathbf{x}) &= \sum_{i \in \mathcal{I}} b_i \widehat{y}_i + \min_{\mathbf{x} \in \mathcal{X}} \sum_{r \in \underline{\mathcal{R}}} \left(c_r - \sum_{i \in \mathcal{I}} a_{ir} \widehat{y}_i \right) x_r \\
&\geq \sum_{i \in \mathcal{I}} b_i \widehat{y}_i + \min_{\mathbf{x} \in \mathcal{X}'} \sum_{r \in \underline{\mathcal{R}}} \overline{c}_r^{\widehat{\mathbf{y}}} x_r \\
&= \sum_{i \in \mathcal{I}} b_i \widehat{y}_i + \overline{c}_j^{\widehat{\mathbf{y}}} + \min_{\substack{x_r \geq 0,\ \forall r \in \underline{\mathcal{R}}_2^>, \\ \sum_{r \in \mathcal{S}^j} x_r \leq 1}} \sum_{r \in \underline{\mathcal{R}}_2^>} \overline{c}_r^{\widehat{\mathbf{y}}} x_r \\
&= \sum_{i \in \mathcal{I}} b_i \widehat{y}_i + \overline{c}_j^{\widehat{\mathbf{y}}} + \min \left\{ 0, \min_{r \in \mathcal{S}^j} \overline{c}_j^{\widehat{\mathbf{y}}} \right\} \\
&= c_j + \langle \mathbf{f}^j, \widehat{\mathbf{y}} \rangle + \min \left\{ 0, \min_{r \in \mathcal{S}^j} \overline{c}_j^{\widehat{\mathbf{y}}} \right\} > ub
\end{aligned}$$

where the first inequality is again because $\overline{c}_r^{\widehat{\mathbf{y}}} \geq 0$ for $r \in \underline{\mathcal{R}}_2^{\leq}$. Finally,

$$\begin{aligned}
z^*|_{x_j=1} &\geq z''|_{x_j=1} = \overline{z}^*|_{\mathbf{x} \in \mathcal{X}} \\
&\geq \max_{\mathbf{y} \in \mathcal{Y}} \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{y}, \mathbf{x}) \geq \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\widehat{\mathbf{y}}, \mathbf{x}) > ub,
\end{aligned}$$

where $\overline{z}^*|_{\mathbf{x} \in \mathcal{X}}$ is the optimal value of $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ when $\mathbf{x}$ is restricted to $\mathcal{X}$. Therefore, any feasible solution to $\mathrm{F}(\underline{\mathcal{R}})$ with x_j equal to one must have an objective value larger than ub, and thus x_j can be removed from the formulation, which completes the proof. □

Remark 2. Applying Proposition 2 requires identifying variables with index in $\underline{\mathcal{R}}_2^>$ that can take a positive value simultaneously with x_j. For VRPs, $\mathcal{S}^j$ can be defined as the set $\{r \in \underline{\mathcal{R}}_2^> : a_{ir} + a_{ij} \leq 1,\ \forall i \in \mathcal{N}\}$, which essentially identifies routes in $\underline{\mathcal{R}}_2^>$ that do not conflict with the given route j while ensuring that each customer is visited only once. It is worth mentioning that if additional information, such as time windows for the CMTVRPTW and battery constraints for electric vehicle (EV) or drone routing (e.g., Desaulniers et al. 2016a, Roberti and Ruthmair 2021), is available to tell that a route $j' \in \underline{\mathcal{R}}_2^>$ is incompatible with route j, then it can be removed from $\mathcal{S}^j$. As a result, more variables might be removed from $\mathrm{F}(\underline{\mathcal{R}})$ because the condition in Proposition 2 becomes easier to satisfy. In addition, the conflict graph constructed in this process can help solve $\mathrm{F}(\underline{\mathcal{R}})$ as an IP at the end. To accelerate each iteration, it is possible to skip computing $\mathcal{S}^j$ exactly and set $\mathcal{S}^j = \underline{\mathcal{R}}_2^>$ instead, which potentially leads to fewer variables being removed each time. Because each iteration is faster now, we can afford to run more iterations and thus find more feasible dual solutions to $\overline{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq})$ for variable fixing.

4.2. Novel LP Formulation for Dual Picking

In this paper, we refer to $\ell_j^{\mathbf{y}} := c_j + \langle \mathbf{f}^j, \mathbf{y} \rangle$ as the Lagrangian underestimate of variable x_j w.r.t. $\mathbf{y}$ and use the number of variables deemed removable by $\boldsymbol{\ell}^{\mathbf{y}} = (\ell_j^{\mathbf{y}})_{j \in \underline{\mathcal{R}}}$ as a measure of the quality of $\mathbf{y}$. Our numerical experiments show that $\boldsymbol{\ell}^{\mathbf{y}}$ significantly differs depending on $\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}$, and thus the quality of $\mathbf{y}$ varies substantially, which aligns with the observation from Sellmann (2004).

4.2.1. High-Level Idea. Finding a $\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}$ of the best quality is NP-hard in general because it involves satisfying the maximum number of linear constraints defined in Propositions 1 and 2, which is equivalent to solving a generalized maximum feasible subsystem problem that is known to be NP-hard (Amaldi and Kann 1995). Nonetheless, finding a single best-quality $\mathbf{y}$ is overkill because we do not have to be restricted to using a single $\mathbf{y}$ for this purpose. In the extreme case, we can solve $\max_{\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}} \langle \mathbf{f}^j, \mathbf{y} \rangle$ for each $j \in \underline{\mathcal{R}}$ to decide individually if x_j can be removed, which requires solving $|\underline{\mathcal{R}}|$ linear programs in total and is polynomial in time complexity. This suggests that we should use multiple $\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}$ to compute different LUs. Now the question becomes how to efficiently obtain multiple $\mathbf{y}$ from $\mathcal{Y}^{\boldsymbol{\pi}}$ that yield high-quality LUs.

Based on the discussion in Section 3.3.1, we propose to iteratively identify a subset $\mathcal{J}$ of $\underline{\mathcal{R}}$ such that the vectors $\mathbf{f}^j$ for $j \in \mathcal{J}$ are close to each other, and then compute $\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}$ that maximizes the inner product $\langle \sum_{j \in \mathcal{J}} \mathbf{f}^j, \mathbf{y} \rangle$. The intuition is that when the vectors $\mathbf{f}^j$ for $j \in \mathcal{J}$ are sufficiently similar, a solution $\mathbf{y}$ maximizing $\sum_{j \in \mathcal{J}} \langle \mathbf{f}^j, \mathbf{y} \rangle$ is likely to also achieve a close-to-maximum value for each individual inner product $\langle \mathbf{f}^j, \mathbf{y} \rangle$, leading to LUs that can potentially eliminate many variables.

4.2.2. Potential Issue and Fix. The unboundedness of $\mathcal{Y}^{\boldsymbol{\pi}}$ implies that there might exist $j \in \mathcal{J}$ such that $\max_{\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}} \langle \mathbf{f}^j, \mathbf{y} \rangle$ goes to $+\infty$. In this case, $d^{\mathcal{J}} := \max_{\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}} \sum_{j \in \mathcal{J}} \langle \mathbf{f}^j, \mathbf{y} \rangle$ is also unbounded. By LP strong duality, it is equivalent to the optimization problem LP^j being infeasible, where LP^j is defined as minimizing $\sum_{r \in \underline{\mathcal{R}}_2^{\leq}} c_r x_r$ subject to $\sum_{r \in \underline{\mathcal{R}}_2^{\leq}} a_{ir} x_r = f_i^j$, $\forall i \in \mathcal{N}$, $\sum_{r \in \underline{\mathcal{R}}_2^{\leq}} a_{kr} x_r \leq f_k^j$, $\forall k \in \mathcal{K}$, and $x_r \geq 0$, $\forall r \in \underline{\mathcal{R}}_2^{\leq}$. By the definition

of $\mathbf{f}^j$, this means no feasible solution can be constructed using variables from $\underline{\mathcal{R}}_2^{\leq}$ when $x_j = 1$, which occurs infrequently in our experiments. A plausible explanation is that all enumerated variables, particularly those in $\underline{\mathcal{R}}_2^{\leq}$, are promising ones because of their relatively small reduced costs. Therefore, the chance that any $j \in \mathcal{J} \subseteq \underline{\mathcal{R}}$ cannot form a feasible solution along with variables in $\underline{\mathcal{R}}_2^{\leq}$ is slim.

However, when LP^j is indeed infeasible for some $j \in \mathcal{J}$, solving $\max_{\mathbf{y}\in\mathcal{Y}^{\boldsymbol{\pi}}} \sum_{j\in\mathcal{J}} \langle \mathbf{f}^j, \mathbf{y}\rangle$ by an LP solver terminates once infeasibility is detected, yielding a possibly low-quality $\widehat{\mathbf{y}} \in \mathcal{Y}^{\boldsymbol{\pi}}$ because of the somewhat arbitrary termination. To address this issue, we only consider bounded $\mathcal{Y}^{\boldsymbol{\pi}}$. More precisely, for $i \in \mathcal{I}$, we lower and upper bound y_i by $-ub$ and ub, respectively, and let $\widehat{\mathcal{Y}^{\boldsymbol{\pi}}} := \mathcal{Y}^{\boldsymbol{\pi}} \cap \{\mathbf{y} \in \mathbb{R}^{|\mathcal{I}|} : -ub \leq y_i \leq ub, \ \forall i \in \mathcal{I}\}$. Note that for VRPs, lower bounding y_i for $i \in \mathcal{I}$ suffices to make $\mathcal{Y}^{\boldsymbol{\pi}}$ bounded because $a_{ir} \in \{0,1\}$, $\forall i \in \mathcal{N}, r \in \underline{\mathcal{R}}_2^{\leq}$. Our dual picking thus involves the following LPs:

$$\widehat{\mathrm{DF}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J}):$$

$$\begin{cases} \max & \displaystyle\sum_{j\in\mathcal{J}} \langle \mathbf{f}^j, \mathbf{y}\rangle \\ \text{s.t.} & \displaystyle\sum_{i\in\mathcal{I}} a_{ir} y_i \leq c_r, \quad \forall r \in \underline{\mathcal{R}}_2^{\leq}, \\ & y_i \geq -ub, \quad \forall i \in \mathcal{I}, \\ & y_i \leq ub, \quad \forall i \in \mathcal{N}, \\ & y_i \leq 0, \quad \forall i \in \mathcal{K}. \end{cases}$$

$$\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J}):$$

$$\begin{cases} \min & \displaystyle\sum_{r\in\underline{\mathcal{R}}_2^{\leq}} c_r x_r + ub \cdot \left(\sum_{i\in\mathcal{I}} w_i + \sum_{i\in\mathcal{N}} v_i \right) \\ \text{s.t.} & \displaystyle\sum_{r\in\underline{\mathcal{R}}_2^{\leq}} a_{ir} x_r + v_i - w_i = \sum_{j\in\mathcal{J}} f_i^j, \quad \forall i \in \mathcal{N}, \\ & \displaystyle\sum_{r\in\underline{\mathcal{R}}_2^{\leq}} a_{kr} x_r - w_i \leq \sum_{j\in\mathcal{J}} f_k^j, \quad \forall k \in \mathcal{K}, \\ & x_r \geq 0, \quad \forall r \in \underline{\mathcal{R}}_2^{\leq}, \\ & w_i \geq 0, \quad \forall i \in \mathcal{I}, \quad v_i \geq 0, \quad \forall i \in \mathcal{N}. \end{cases}$$

We choose to work with $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ instead of $\widehat{\mathrm{DF}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ for implementation simplicity and computational efficiency. First of all, $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ can be modified from $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ more easily than $\widehat{\mathrm{DF}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ inside a solver. Moreover, the dual simplex method has been empirically demonstrated to be superior to the primal simplex method (Bixby 2002). State-of-the-art LP solvers, such as Gurobi and CPLEX, almost always apply the dual simplex method in the default setting when running with a single thread. We iteratively vary the set $\mathcal{J}$ and solve $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ by the dual simplex to obtain an optimal dual solution $\widehat{\mathbf{y}}$, which is subsequently used to compute LUs and identify the removable variables as per Propositions 1 and 2. Notably, it suffices to modify the right-hand side of $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \mathcal{J})$ when $\mathcal{J}$ is changed. Furthermore, the revised LP can be solved fast because of the warm-start effect of the dual simplex method in this case.

4.2.3. Further Discussion. Changing $\mathcal{Y}^{\boldsymbol{\pi}}$ into $\widehat{\mathcal{Y}^{\boldsymbol{\pi}}}$ narrows the search region, which can potentially deteriorate the quality of $\widehat{\mathbf{y}}$ obtained. However, according to our numerical experiments, such a presumed side effect is negligible. To provide some intuition for this observation, let us consider the formulation with $\mathcal{J} = \{j\}$, denoted by $\widehat{\mathrm{DF}}(\underline{\mathcal{R}}_2^{\leq}, j)$, and its dual LP, denoted by $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, j)$, for a given $j \in \underline{\mathcal{R}}$. Let $\widehat{d}^{\{j\}}$ be the optimal value of $\widehat{\mathrm{DF}}(\underline{\mathcal{R}}_2^{\leq}, j)$. Suppose $d^{\{j\}} := \max_{\mathbf{y}\in\mathcal{Y}^{\boldsymbol{\pi}}} \langle \mathbf{f}^j, \mathbf{y}\rangle > ub$, then there exists $\mathbf{y} \in \mathcal{Y}^{\boldsymbol{\pi}}$ certifying that variable x_j can be removed. Let $(\mathbf{x}^*, \mathbf{w}^*, \mathbf{v}^*)$ be an optimal solution to $\widehat{\mathrm{F}}(\underline{\mathcal{R}}_2^{\leq}, \{j\})$. When $\sum_{i\in\mathcal{I}} w_i^* + \sum_{i\in\mathcal{N}} v_i^* = 0$, we have that $\mathbf{x}^*$ is feasible to LP^j, and thus, according to strong duality, it holds that $\widehat{d}^{\{j\}} \geq d^{\{j\}} > ub$. If $\sum_{i\in\mathcal{I}} w_i^* + \sum_{i\in\mathcal{N}} v_i^* \geq 1$, then again, we have $\widehat{d}^{\{j\}} > ub$ when $c_r > 0$, suggesting that $\widehat{\mathcal{Y}^{\boldsymbol{\pi}}}$ still contains some elements which can certify that variable x_j is removable. In this sense, changing $\mathcal{Y}^{\boldsymbol{\pi}}$ to $\widehat{\mathcal{Y}^{\boldsymbol{\pi}}}$ causes a minimum difference. It is worth noticing that the above two cases (i.e., $\sum_{i\in\mathcal{I}} w_i^* + \sum_{i\in\mathcal{N}} v_i^* \leq 0$ or $\sum_{i\in\mathcal{I}} w_i^* + \sum_{i\in\mathcal{N}} v_i^* \geq 1$) are likely to happen because, for many problems including VRPs and CRPs, we have $f_i^j \in \mathbb{Z}$ for $i \in \mathcal{I}$.

4.3. Overview of DeLuxing

Algorithm 1 outlines the DeLuxing method, which consists of three steps explained in detail in Section 5. It is controlled by three input parameters: the number of clusters p and two threshold constants β_1 and β_2. In Step 1, an optimal dual solution to $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ is first obtained to compute reduced costs and initialize the index sets. In Step 2, the index set $\underline{\mathcal{R}}$ is first partitioned into p clusters via either the k-means++ clustering method (Arthur and Vassilvitskii 2007) or a simple but effective heuristic approach. In Step 3, a deep search for qualified dual solutions of good quality is performed using the centroid of each cluster as an initial reference point. This process involves calling the subroutine Algorithm 2 repeatedly with refined reference points, whose correctness is guaranteed by Propositions 1 and 2. Finally, Algorithm 1 outputs the index set of all variables certified as removable.

Algorithm 1 (The Deep Lagrangian Underestimate Fixing (DeLuxing) Algorithm)

Input: The number of clusters p, two threshold constants β_1 and β_2.

Step 1. *Initialization:* Solve $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ and obtain an optimal dual solution $\boldsymbol{\pi}$. Set $\underline{\mathcal{R}} \leftarrow \{r \in \underline{\mathcal{R}} : \overline{c}_r^{\boldsymbol{\pi}} \le g\}$, $\mathcal{R}_1 \leftarrow \left\{r \in \underline{\mathcal{R}} : \overline{c}_r^{\boldsymbol{\pi}} \le \frac{g}{2}\right\}$, $\mathcal{R}_2 \leftarrow \underline{\mathcal{R}} \setminus \mathcal{R}_1$, and $\mathcal{H} \leftarrow \emptyset$.

Step 2. *Clustering:*

If $|\underline{\mathcal{R}}| \le \beta_1$

Apply the k-means++ method to partition $\underline{\mathcal{R}}$ into p clusters, $\underline{\mathcal{R}}^1, \ldots, \underline{\mathcal{R}}^p$.

Else

Apply the ClustByNorm heuristic to partition $\underline{\mathcal{R}}$ into p clusters, $\underline{\mathcal{R}}^1, \ldots, \underline{\mathcal{R}}^p$.

Step 3. *Deep Search:*

For $i = 1$ to p

Set $\tilde{\mathcal{J}} \leftarrow \underline{\mathcal{R}}^i \setminus \mathcal{H}$.

Do

Call the subroutine with input $\tilde{\mathcal{J}}$, $\mathcal{R}_1$, $\mathcal{R}_2$, and obtain the output $\mathcal{D}$.

Set $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{D}$, $\tilde{\mathcal{J}} \leftarrow \mathcal{D} \setminus \tilde{\mathcal{J}}$, $\mathcal{R}_1 \leftarrow \mathcal{R}_1 \setminus \mathcal{H}$, and $\mathcal{R}_2 \leftarrow \mathcal{R}_2 \setminus \mathcal{H}$.

While $|\mathcal{D}| \ge \beta_2$

End

Output: The index set $\mathcal{H}$.

Algorithm 2 (The Subroutine in DeLuxing)

Input: Three index sets $\tilde{\mathcal{J}}$, $\mathcal{R}_1$, and $\mathcal{R}_2$.

Substep 1. Solve $\widehat{\mathrm{F}}(\mathcal{R}_1, \tilde{\mathcal{J}})$ and obtain an optimal dual solution $\widehat{\mathbf{y}}$.

Substep 2. Compute $\mathcal{D} \leftarrow \{j \in \mathcal{R}_2 : \langle \mathbf{f}^j, \widehat{\mathbf{y}} \rangle > ub - c_j\} \cup \{j \in \mathcal{R}_1 : \langle \mathbf{f}^j, \widehat{\mathbf{y}} \rangle > ub - c_j - \min\{\eta_j, 0\}\}$, where $\eta_j = \min_{r \in \mathcal{S}^j}\{c_r - \langle \widehat{\mathbf{y}}, \mathbf{a}_r \rangle\}$ and $\mathcal{S}^j = \{r \in \mathcal{R}_2 : x_r \text{ is compatible with } x_j\}$.

Output: The index set $\mathcal{D}$.

5. Elaboration on Every Step of DeLuxing

5.1. Step 1: Initialization

In this step, we first solve the linear program $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ to obtain an optimal dual solution $\boldsymbol{\pi}$. Then $\boldsymbol{\pi}$ is used to initialize two index sets $\mathcal{R}_1$ and $\mathcal{R}_2$, which keep track of the columns with reduced cost no larger than half of the current gap g and those within $(\frac{g}{2}, g]$ w.r.t. $\boldsymbol{\pi}$, respectively. It is worth noting that the dual solution used to enumerate $\underline{\mathcal{R}}$, referred to as $\tilde{\boldsymbol{\pi}}$, can also be used for this purpose. However, we compute a new $\boldsymbol{\pi}$ because it updates the reduced costs and can help to remove some variables. We observe in our numerical experiments that, on average, about 10% of the enumerated variables can be certified to be removable using the updated reduced costs, that is, $|\{r \in \underline{\mathcal{R}} : \overline{c}_r^{\boldsymbol{\pi}} > g\}| \approx 10\% \times |\underline{\mathcal{R}}|$. To improve computational efficiency, when the cardinality of $\underline{\mathcal{R}}$ is in the millions or higher, we skip solving $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ and directly set $\boldsymbol{\pi} = \tilde{\boldsymbol{\pi}}$ to initialize $\mathcal{R}_1$ and $\mathcal{R}_2$.

5.2. Step 2: Clustering

We maximize the inner product $\langle \sum_{j \in \mathcal{J}} \mathbf{f}^j, \mathbf{y} \rangle = |\mathcal{J}| \langle \overline{\mathbf{f}}, \mathbf{y} \rangle$ in the hope that the resulting $\mathbf{y}$ achieves close-to-optimal value for each individual $\langle \mathbf{f}^j, \mathbf{y} \rangle$, where $\overline{\mathbf{f}} = \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathbf{f}^j$. Using the Cauchy-Schwarz inequality, we can derive $|\langle \mathbf{f}^j, \mathbf{y} \rangle - \langle \overline{\mathbf{f}}, \mathbf{y} \rangle| \le \|\mathbf{f}^j - \overline{\mathbf{f}}\| \|\mathbf{y}\|$, which suggests we are likely to achieve our goal as long as $\|\mathbf{f}^j - \overline{\mathbf{f}}\|$ is small. This naturally leads us to the well-known k-means clustering problem that seeks to partition n observations $\{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^n\}$ in d dimension into k clusters $C_1, C_2, \ldots, C_k$ to minimize the within-cluster sum of squares, defined as $\sum_{i=1}^{k} \sum_{\mathbf{u} \in C_i} \|\mathbf{u} - \boldsymbol{\mu}^i\|^2$, where $\boldsymbol{\mu}^i$ is the mean (also called centroid) of points in the i-th cluster C_i.

Although finding the optimal solution to the k-means clustering problem in d dimension is NP-hard even for two clusters (Aloise et al. 2009), many effective heuristics are available such as the Lloyd algorithm (Lloyd 1982), refinement with Bradley and Fayyad's initialization (Bradley and Fayyad 1998), and the k-means++ (Arthur and Vassilvitskii 2007). Because k-means++ is known for its generally good performance (Celebi et al. 2013) and easy implementation, it has been used as the default method for determining initial cluster centroid positions in the "kmeans" function of MATLAB. We also choose to use it in our C++ implementation.

We observe in our experiments that clustering vectors $\{\widehat{\mathbf{f}}^j\}_{j \in \underline{\mathcal{R}}}$ instead of $\{\mathbf{f}^j\}_{j \in \underline{\mathcal{R}}}$ significantly improves the speed while yielding clusters that achieve nearly the same or sometimes even better overall performance for Algorithm 1. Here, $\widehat{\mathbf{f}}^j := (\mathbf{f}^j_i)_{i \in \mathcal{N}}$ is a subvector of $\mathbf{f}^j$ that includes only the dimensions in $\mathcal{N}$. A possible explanation for this observation is that when two columns $j_1, j_2 \in \underline{\mathcal{R}}$ have similar coefficients a_{ij_1} and a_{ij_2} for $i \in \mathcal{N}$, it is likely that a_{ij_1} and a_{ij_2} are also close for $i \in \mathcal{K}$ because they correspond to coefficients of cutting planes. As a result, $\widehat{\mathbf{f}}^j$ serves as a good representation of $\mathbf{f}^j$ for the purpose of clustering. In our implementation, we cluster $\{\widehat{\mathbf{f}}^j\}_{j \in \underline{\mathcal{R}}}$. However, readers are encouraged to explore alternative options that may be more suitable for their specific problems.

5.2.1. The ClustByNorm Heuristic.

Although we can easily parallelize the computation using OpenMP (a library for parallel programming that supports C, C++, and Fortran) to achieve significant acceleration, the clustering process can still be time-consuming when the size of $\underline{\mathcal{R}}$ is in the millions. In such cases, we propose to use a simple but surprisingly effective heuristic, which we call ClustByNorm, to perform the clustering in place of the k-means++ method. It starts with computing the l_2 norm of each vector $\|\mathbf{f}^j\|$ for $j \in \underline{\mathcal{R}}$ and sorts them in non-increasing order. Then, we partition the sorted list into p clusters, each containing roughly $q = \lfloor |\underline{\mathcal{R}}|/p \rfloor$ vectors.

Specifically, we assign the $(k-1)*q+1$ to $(k*q)$-th vectors in the sorted list to the k-th cluster for $k=1,\ldots,p-1$, and the remaining vectors to the p-th cluster. Although ClustByNorm is slightly less effective than the k-means++ method in terms of the resulting DeLuxing's capability to remove columns, it leads to significant speedup when $|\underline{\mathcal{R}}|$ is large. We provide a detailed comparison of the performance of k-means++ and ClustByNorm in Section 7.1. By default, we use the k-means++ method for clustering and switch to ClustByNorm when $|\underline{\mathcal{R}}|$ exceeds a threshold constant β_1.

5.3. Step 3: Deep Search

For each cluster i, we first update it by $\tilde{\mathcal{J}} \leftarrow \underline{\mathcal{R}}^i \setminus \mathcal{I}$ to exclude those identified as removable. Then its centroid $\boldsymbol{\mu} := \frac{1}{|\tilde{\mathcal{J}}|}\sum_{j\in\tilde{\mathcal{J}}}\mathbf{f}^j$ is used as a reference point to start the search. Specifically, we try to find a $\mathbf{y}$ that maximizes $\langle\boldsymbol{\mu},\mathbf{y}\rangle$, which is accomplished by solving $\widehat{\mathrm{F}}(\mathcal{R}_1,\tilde{\mathcal{J}})$ using the subroutine Algorithm 2. It returns the index set of removable variables $\mathcal{D}$ with the given input. However, bundling elements in a cluster according to this one-time clustering may not achieve the most desirable result. One reason is that the conditions in Propositions 1 and 2 aim to satisfy inequalities, whereas the clustering only concerns part of the inequalities; that is, it tries to maximize the inner product on the left-hand side. Adding one extra dimension with a value of c_j to each vector $\mathbf{f}^j$ and clustering the updated vectors do not provide noticeable improvements, indicating the difficulty of incorporating information from the right-hand side. Moreover, the k-means++ method or the ClustByNorm is not perfect and is not likely to yield the best clusters.

The proposed deep search tries to address this concern. Essentially, it iteratively builds an artificial cluster by utilizing the most recently identified set $\mathcal{D}$ excluding $\tilde{\mathcal{J}}$ (i.e., those that were used as input to Algorithm 2 to generate this $\mathcal{D}$). To the best of our knowledge, this idea is new in the literature. The rationale behind this approach is that the elements in a set $\mathcal{D}$ correspond to variables deemed removable by a common dual solution, which implicitly considers the whole inequalities and captures hidden similarities that might have been missed by the initial clustering. We remove $\tilde{\mathcal{J}}$ from $\mathcal{D}$ because its information has already been used to generate $\mathcal{D}$ and is likely to be redundant and can adversely impact the next iteration. Multiple high-quality dual solutions are effectively picked in the do-while loop, and the total computational effort can be easily controlled by the input threshold constant β_2. Specifically, the total number of calls to the subroutine is upper bounded by $(p+\lceil|\underline{\mathcal{R}}|/\beta_2\rceil)$ because, by design, all the index sets $\mathcal{D}$ produced are nonoverlapping.

6. Extensions

In this section, we expand upon the fundamental concept underlying DeLuxing to a broader range of contexts. Firstly, based on this concept, we prove that many integer variables can be relaxed into continuous ones in the IP solved to close a BBN, resulting in significant acceleration when the IP is large-scale and difficult to solve. Additionally, we demonstrate the concept can also be applied to enhance cutting plane addition. Furthermore, we propose an effective primal heuristic in which DeLuxing plays a crucial role. The effectiveness of these extensions is demonstrated individually using instances of the CMTVRPTW in Section 7.2. Moreover, we present the acceleration achieved by the new primal heuristic for solving the CVRP and VRPTW in Section 7.5. The first two extensions are not effective for the CVRP and VRPTW because a large number of relatively easy LPs and IPs are solved therein, whereas these extensions focus on dealing with a few more challenging LPs and IPs. Nonetheless, we believe the ideas behind them can potentially be applied to general MIP settings.

6.1. Variable Relaxation

Solving $\mathrm{F}(\underline{\mathcal{R}})$ as an IP by a solver is a convenient and effective way to close a BBN. The computational difficulty of $\mathrm{F}(\underline{\mathcal{R}})$ relies heavily on the number of integer variables, and it usually takes many rounds of branching and cutting plane additions before reducing the size of $\underline{\mathcal{R}}$ to a manageable level (e.g., Pessoa et al. (2020) requires $|\underline{\mathcal{R}}| \le 10{,}000$). Relaxing the integer requirement for a large portion of variables is expected to bring substantial acceleration. If such relaxation is allowed, $\mathrm{F}(\underline{\mathcal{R}})$ can be solved as a MIP at a much earlier stage in the BPC method, saving a considerable amount of computational effort on branching and adding cutting planes to reduce the size of $\underline{\mathcal{R}}$. The following Proposition 3 indicates that in some cases, we can relax x_r for $r \in \underline{\mathcal{R}}_2^{>}$ in $\mathrm{F}(\underline{\mathcal{R}})$ as continuous variables without compromising optimality. Let $\check{\mathrm{F}}(\underline{\mathcal{R}})$ be the formulation obtained by relaxing variables x_r for $r \in \underline{\mathcal{R}}_2^{>}$ in $\mathrm{F}(\underline{\mathcal{R}})$ to be continuous and adding the constraint $\sum_{r\in\underline{\mathcal{R}}_2^{>}} x_r \le 1$. Let $\mathcal{P} \subset \mathbb{R}_+^{|\underline{\mathcal{R}}|}$ be the polyhedron corresponding to the feasible region of $\check{\mathrm{F}}(\underline{\mathcal{R}})$, $\mathcal{E} \subset \mathcal{P}$ be the set of extreme points of $\mathcal{P}$, and $\mathcal{X}^* \subset \mathcal{P}$ be the set of optimal solutions to $\check{\mathrm{F}}(\underline{\mathcal{R}})$.

Proposition 3. *If in* $\mathrm{F}(\underline{\mathcal{R}})$, $a_{ir} \in \{0,1\}$ *and* $b_i \in \mathbb{Z}$ $\forall i \in \mathcal{N}$, $r \in \underline{\mathcal{R}}$, *then it follows that* $\mathcal{X}^* \cap \mathcal{E} \subset \mathbb{Z}_+^{|\underline{\mathcal{R}}|}$.

Proof. Let us consider any $\overline{\mathbf{x}} \in \mathcal{X}^* \cap \mathcal{E}$. It holds that $\overline{x}_r \in \mathbb{Z}_+$ for $r \in \underline{\mathcal{R}}_2^{\le}$ because of the integer requirement in $\check{\mathrm{F}}(\underline{\mathcal{R}})$. With the coefficients $a_{ir} \in \{0,1\}$ and $b_i \in \mathbb{Z}$ for all $i \in \mathcal{N}$ and $r \in \underline{\mathcal{R}}$, it follows that $\sum_{r\in\underline{\mathcal{R}}_2^{\le}} a_{ir}\overline{x}_r \in \mathbb{Z}$ $\forall i \in \mathcal{N}$. Let $u_i := \sum_{r\in\underline{\mathcal{R}}_2^{>}} a_{ir}\overline{x}_r$. Consequently, $u_i = b_i - \sum_{r\in\underline{\mathcal{R}}_2^{\le}} a_{ir}\overline{x}_r$ is integral for all $i \in \mathcal{N}$. For $r \in \underline{\mathcal{R}}_2^{>}$, $\overline{x}_r$ is nonnegative, thus $u_i \in \mathbb{Z}_+$. Additionally, the constraint $\sum_{r\in\underline{\mathcal{R}}_2^{>}} x_r \le 1$ in $\check{\mathrm{F}}(\underline{\mathcal{R}})$ implies $\sum_{r\in\underline{\mathcal{R}}_2^{>}} \overline{x}_r \le 1$. Let $\mathcal{Q} := \{r \in \underline{\mathcal{R}}_2^{>} : 0 < \overline{x}_r < 1\}$. If $\mathcal{Q}$ is an empty set, no further proof is needed. Let $\mathcal{N}^r := \{i \in \mathcal{N} : a_{ir} = 1\}$ for $r \in \mathcal{Q}$ and $\tilde{\mathcal{N}} := \cup_{r\in\mathcal{Q}}\mathcal{N}^r$.

Claim 1. We claim that if $\mathcal{Q} \neq \emptyset$, then $\mathcal{N}^{r_1} = \mathcal{N}^{r_2}$ $\forall r_1, r_2 \in \mathcal{Q}$.

The claim is proved by contradiction. First, $\mathcal{Q} \neq \emptyset$ and $\sum_{r \in \underline{\mathcal{R}}_2^>} \overline{x}_r \leq 1$ imply there does not exist $r \in \underline{\mathcal{R}}_2^>$ such that $\overline{x}_r \geq 1$. Thus, we have $x_r = 0 \ \forall r \in \underline{\mathcal{R}}_2^> \setminus \mathcal{Q}$. Suppose there exist $r_1, r_2 \in \mathcal{Q}$ such that $\mathcal{N}^{r_1} \neq \mathcal{N}^{r_2}$. As a result, there exist $k \in \mathcal{N}$, $r', r'' \in \mathcal{Q}$ such that $k \in \mathcal{N}^{r'}$ and $k \notin \mathcal{N}^{r''}$. Therefore, we have $0 < x_{r'} \leq \sum_{r \in \underline{\mathcal{R}}_2^>} a_{kr} \overline{x}_r = \sum_{r \in \mathcal{Q}} a_{kr} \overline{x}_r < \sum_{r \in \mathcal{Q}} \overline{x}_r = \sum_{r \in \underline{\mathcal{R}}_2^>} \overline{x}_r \leq 1$. This implies $u_k \in (0,1)$, which contradicts the fact that u_k is an integer and thus proves the claim. Next we will show that if $\mathcal{Q} \neq \emptyset$, then $\overline{\mathbf{x}} \notin \mathcal{E}$.

Note that $\mathcal{Q}$ cannot be a singleton because if $\mathcal{Q} = \{r\}$, then $0 < \sum_{r \in \underline{\mathcal{R}}_2^>} a_{ir} \overline{x}_r = \overline{x}_r < 1$ for $i \in \mathcal{N}^r$, which again contradicts the fact that u_i is integral. Consider any two distinct $r_1, r_2 \in \mathcal{Q}$. According to the claim, we have $\mathcal{N}^{r_1} = \mathcal{N}^{r_2}$, that is, $a_{ir_1} = a_{ir_2}$ for all $i \in \mathcal{N}$. Let $\tilde{\mathbf{x}}' = (\tilde{x}'_r)_{r \in \underline{\mathcal{R}}}$ and $\tilde{\mathbf{x}}'' = (\tilde{x}''_r)_{r \in \underline{\mathcal{R}}}$ be set to $\tilde{x}'_r = \tilde{x}''_r = \overline{x}_r$ for $r \in \underline{\mathcal{R}} \setminus \{r_1, r_2\}$ and $\tilde{x}'_{r_1} = \overline{x}_{r_1} - \epsilon$, $\tilde{x}'_{r_2} = \overline{x}_{r_2} + \epsilon$, $\tilde{x}''_{r_1} = \overline{x}_{r_1} + \epsilon$, and $\tilde{x}''_{r_2} = \overline{x}_{r_2} - \epsilon$, where $\epsilon > 0$ is small enough to ensure $\tilde{x}'_{r_1}$ and $\tilde{x}''_{r_2}$ are nonnegative. Then $\sum_{r \in \underline{\mathcal{R}}} a_{ir} \overline{x}_r = \sum_{r \in \underline{\mathcal{R}}} a_{ir} \tilde{x}'_r = \sum_{r \in \underline{\mathcal{R}}} a_{ir} \tilde{x}''_r$ and thus $\tilde{\mathbf{x}}'$ and $\tilde{\mathbf{x}}''$ are feasible solutions to $\check{\mathrm{F}}(\underline{\mathcal{R}})$. Furthermore, it follows that $\overline{\mathbf{x}} = (\tilde{\mathbf{x}}' + \tilde{\mathbf{x}}'')/2$, suggesting that $\overline{\mathbf{x}}$ is not an extreme point of $\mathcal{P}$, that is, $\overline{\mathbf{x}} \notin \mathcal{E}$.

Therefore, the set $\mathcal{Q}$ has to be empty when $\overline{\mathbf{x}} \in \mathcal{E}$. The fact that $\sum_{r \in \underline{\mathcal{R}}_2^>} x_r \leq 1$ guarantees that $0 \leq x_r \leq 1$. Consequently, all the elements of $\overline{\mathbf{x}}$ take an integer value, which completes the proof. □

Remark 3. Proposition 3 guarantees that when the constraint coefficients a_{ir} are binaries and b_i are integers, any optimal solution to $\check{\mathrm{F}}(\underline{\mathcal{R}})$ is also integral as long as it is an extreme point of the underlying polyhedron. For set partitioning formulations of VRPs and CRPs, the coefficients $a_{ir} \in \{0,1\}$ and $b_i = 1$ for all $i \in \mathcal{N}$ and $r \in \underline{\mathcal{R}}$, and thus the conditions are satisfied. However, it should be noted that when there exist two distinct $r_1, r_2 \in \underline{\mathcal{R}}$ such that $c_{r_1} = c_{r_2}$ and $a_{ir_1} = a_{ir_2} \ \forall i \in \mathcal{N}$, it is possible for $\check{\mathrm{F}}(\underline{\mathcal{R}})$ to have an optimal solution that is not integral. This means there could be two identical columns that cannot be deleted because of the absence of certain feasibility requirements in the formulation, which is added as lazy cuts during the solution process. For the standard CVRP and VRPTW, this does not occur because there are no missing feasibility requirements in their formulations. In the case of CMTVRPTW, where the superstructure feasibility constraints are initially absent and are added dynamically using a callback function, such a scenario can happen. These constraints decide if there exists a feasible schedule to execute the selected vehicle routes by a given fleet, allowing each vehicle to make multiple trips (refer to section 3.2 of Yang 2023 for details). Nonetheless, as long as the solver returns an optimal solution that represents an extreme point of $\mathcal{P}$, the proposed relaxation in Proposition 3 can still be applied without sacrificing optimality. Note that modern MIP solvers may yield an optimal solution that is not necessarily an extreme point via some primal heuristics. In this case, a callback function that cuts off such solutions by lazy constraints (so-called no-good cuts, Hooker et al. 1999) can be used to guarantee correctness.

The proposed relaxation can be performed even more aggressively with the help of a simple search and some lazy constraints to ensure optimality and integrality. Let $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$ be the formulation obtained by relaxing variables x_r for $r \in \underline{\mathcal{R}}_3^>$ in the original formulation $\mathrm{F}(\underline{\mathcal{R}})$ to be continuous and adding the constraints $\sum_{r \in \underline{\mathcal{R}}_2^>} x_r \leq 1$ and $\sum_{r \in \underline{\mathcal{R}}_3^>} x_r \leq 2$. We solve $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$ by a MIP solver with the callback function presented in Algorithm 3 that adds lazy constraints.

To ease the presentation, given a feasible solution $\overline{\mathbf{x}}$ to $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$, we define $\mathcal{R}^f := \{r \in \underline{\mathcal{R}}_3^> : \ \overline{x}_r > 0\}$, $\mathcal{R}^t := \{r \in \underline{\mathcal{R}} \setminus \underline{\mathcal{R}}_3^> : \ \overline{x}_r > 0\}$, $\tilde{c} := \sum_{r \in \mathcal{R}^t} c_r \overline{x}_r$, and $\tilde{\mathbf{b}} := (\tilde{b}_i)_{i \in \mathcal{N}}$, where $\tilde{b}_i = b_i - \sum_{r \in \mathcal{R}^t} a_{ir} \overline{x}_r$. The callback function is triggered whenever such a feasible solution $\overline{\mathbf{x}}$ is identified. It first examines whether all $\overline{x}_r$ values for $r \in \underline{\mathcal{R}}_3^>$ are integers. If they are, no further action is required, and the callback function terminates, returning control to the solver. If any $\overline{x}_r$ for $r \in \underline{\mathcal{R}}_3^>$ is not an integer, the callback function proceeds by searching for a feasible solution better than the one achieving the current upper bound (known as the incumbent). Specifically, it checks whether any two columns $r' \neq r''$ and $r', r'' \in \mathcal{R}^f$, together with columns with indices in $\mathcal{R}^t$ that have been selected an integral number of times in $\overline{\mathbf{x}}$, can form a superior feasible solution. This search can be conducted in a brute-force fashion, as we only need to consider $|\mathcal{R}^f|^2$ combinations. Notably, the size of $\mathcal{R}^f$ is small, typically less than a few dozen. After the search, a lazy constraint $\sum_{r \in \mathcal{R}^f \cup \mathcal{R}^t} x_r \leq |\mathcal{R}^t| + 1$ is added and the callback terminates.

Algorithm 3 (The Callback Function)

Input: A feasible solution $\overline{\mathbf{x}}$ to $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$ and current upper bound $\tilde{z}$

if $\exists r \in \underline{\mathcal{R}}_3^>$ *such that* $\overline{x}_r \in \mathcal{R}_+ \setminus \mathbb{Z}_+$ **then**

Step 1. Search for $r', r'' \in \mathcal{R}^f$, $r' \neq r''$ with the smallest $c_{r'} + c_{r''}$ such that $\mathbf{a}_{r'} + \mathbf{a}_{r''} = \tilde{\mathbf{b}}$ and $c_{r'} + c_{r''} < \tilde{z} - \tilde{c}$. If one is found, update the incumbent solution to $\tilde{\mathbf{x}}$ by Equation (3).

Step 2. Add a lazy constraint $\sum_{r \in \mathcal{R}^f \cup \mathcal{R}^t} x_r \leq |\mathcal{R}^t| + 1$.

$\tilde{\mathbf{x}} := (\tilde{x}_r)_{r \in \underline{\mathcal{R}}}$, where

$$\tilde{x}_r = \begin{cases} 1, & \text{if } r \in \mathcal{R}^t \cup \{r', r''\}, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The following Proposition 4 guarantees that under some mild conditions, we can obtain an optimal

solution to $\mathrm{F}(\underline{\mathcal{R}})$ by solving $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$ with the proposed callback function in Algorithm 3.

Proposition 4. *Suppose in* $\mathrm{F}(\underline{\mathcal{R}})$, *all variables* x_r *for* $r \in \underline{\mathcal{R}}$ *are required to be binary,* $a_{ir} \in \{0,1\}$, *and* $b_i \in \mathbb{Z}\ \ \forall i \in \mathcal{N}$, $r \in \underline{\mathcal{R}}$. *Solving* $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$ *by an MIP solver that is equipped with the callback function described in Algorithm 3 and can find an optimal solution corresponding to an extreme point of the polyhedron (i.e., the feasible region of* $\tilde{\mathrm{F}}(\underline{\mathcal{R}})$*) guarantees to find an optimal solution to* $\mathrm{F}(\underline{\mathcal{R}})$.

Proof. If all $\overline{x}_r$ values for $r \in \underline{\mathcal{R}}_3^{\geq}$ are integers, then $\overline{\mathbf{x}}$ is feasible to $\mathrm{F}(\underline{\mathcal{R}})$. We only need to consider the case that there exists $\overline{x}_r$ taking a fractional value for some $r \in \underline{\mathcal{R}}_3^{>}$. Because $\sum_{r\in\mathcal{R}^f} x_r \leq \sum_{r\in\underline{\mathcal{R}}_3^{>}} x_r \leq 2$ and $x_r \in \{0,1\}$ for $r \in \mathcal{R}^t \subseteq \underline{\mathcal{R}}$, we have $\sum_{r\in\mathcal{R}^f\cup\mathcal{R}^t} x_r = \sum_{r\in\mathcal{R}^t} x_r + \sum_{r\in\mathcal{R}^f} x_r \leq |\mathcal{R}^t| + 2$. Because $\sum_{r\in\mathcal{R}^f\cup\mathcal{R}^t} x_r \in \mathbb{Z}_+$, the lazy constraint $\sum_{r\in\mathcal{R}^f\cup\mathcal{R}^t} x_r \leq |\mathcal{R}^t| + 1$ will eliminate part of the feasible region with $\sum_{r\in\mathcal{R}^f\cup\mathcal{R}^t} x_r = |\mathcal{R}^t| + 2$. We claim that if an optimal solution exists in this eliminated part, it can be found, and thus, optimality can still be guaranteed.

Note that $\sum_{r\in\mathcal{R}^f\cup\mathcal{R}^t} x_r = |\mathcal{R}^t| + 2$ implies $x_r = 1$ for all $r \in \mathcal{R}^t$ and $\sum_{r\in\mathcal{R}^f} x_r = 2$. Thus, we only need to search for all $r', r'' \in \mathcal{R}^f$ and $r' \neq r''$, which, when combined with columns indexed by $r \in \mathcal{R}^t$, can form a superior feasible solution to the current incumbent solution. As shown in Algorithm 3, the callback function searches all such qualified pairs of columns and picks the best one, which completes the claim. It remains to show that integrality is also guaranteed.

It suffices to show the lazy constraints added by the callback can prevent any fractional solution from being considered feasible. We only need to consider fractional solutions $\overline{\mathbf{x}}$ with $\sum_{r\in\mathcal{R}^f} \overline{x}_r \leq 1$. By a similar argument to that provided in the above proof of Proposition 3, we can show that such $\overline{\mathbf{x}}$ cannot be an extreme point of the corresponding polyhedron, which completes the proof. □

6.2. A New Way of Cutting Plane Addition

Cutting planes play a key role in modern branch-and-cut and BPC methods, which iteratively improve the dual bound and thus close the optimality gap. After obtaining an optimal solution to the current LP relaxation, cut separators are employed to identify violated valid inequalities. These inequalities can cut off the current fractional LP solution and are then incorporated as constraints in the LP. This process continues until some termination criteria are met.

For BPC methods, solving each LP after each round of cut addition requires repeated CG. When an enumerated pool is available, CG can be performed through inspection, which is much more efficient compared with a labeling algorithm. Nevertheless, this process can still be time-consuming when a considerable number of rounds of cut addition and CG are performed to solve the LP. Inspired by DeLuxing, we propose a new approach to streamline this procedure. Specifically, instead of starting with the columns currently present in the LP, we first include a moderate-sized (typically no larger than 50,000) subset of columns, denoted by $\tilde{\mathcal{R}}$, from the enumerated pool $\underline{\mathcal{R}}$. We then obtain formulation $\overline{\mathrm{F}}(\tilde{\mathcal{R}})$, with $\underline{\mathcal{R}}$ in formulation $\overline{\mathrm{F}}(\underline{\mathcal{R}})$ from Section 3.2 replaced by $\tilde{\mathcal{R}}$. One particular selection criterion that works well numerically is to include all columns from the enumerated pool with reduced costs not surpassing half of the gap, that is, $\tilde{\mathcal{R}} = \underline{\mathcal{R}}_2^{\leq}$. Subsequently, we generate cuts, such as subset-row inequalities (Jepsen et al. 2008) and rounded capacity cuts (Laporte and Nobert 1983), to tighten $\overline{\mathrm{F}}(\tilde{\mathcal{R}})$ and directly resolve it without introducing new columns from the remaining pool. This process is repeated until tailing off or hitting prespecified limits. Before termination, all remaining columns in the pool are introduced into the tightened $\overline{\mathrm{F}}(\tilde{\mathcal{R}})$ to compute the final lower bound, ensuring its validity.

This approach presents two notable advantages. Firstly, because of the significantly smaller size of $\tilde{\mathcal{R}}$ compared with $\underline{\mathcal{R}}$, we can circumvent the need to solve large LPs when CG adds an excessive number of columns. Secondly, when $\tilde{\mathcal{R}}$ is selected such that $\underline{\mathcal{R}}_2^{\leq} \subseteq \tilde{\mathcal{R}}$, additional opportunities for variable fixing emerge. More precisely, we can directly apply Propositions 1 and 2 to remove columns whenever the LP is resolved and a new dual solution is obtained. We acknowledge that inexact objective values are computed in the process because only a subset of columns are included in the current LP, which affects the cuts generated and potentially the quality of the lower bound obtained at the termination of the procedure. However, we emphasize the validity of the final lower bound, as all remaining columns in the pool will be added back to compute the bound. Our numerical experiments demonstrate that adding cuts in this manner yields almost identical bounds in our numerical experiments and results in significant speedup for some instances, as shown in Section 7.2.

6.3. An Effective Primal Heuristic

Using a solver to solve the current RMP as an IP serves as a commonly employed primal heuristic within the BPC framework (Pessoa et al. 2020). The success of this approach is closely tied to the number of columns present in the RMP. An excessive number of columns can result in long computation times, whereas too few columns may produce feasible solutions of poor quality or result in infeasibility. To tackle this challenge, a straightforward approach is to only keep in the IP the smallest $\widehat{\beta}$ columns in terms of reduced costs, where $\widehat{\beta}$ is a constant. This approach leaves room for improvement, especially for the CMTVRPTW and its variants.

Similar to Pessoa et al. (2009), Marques et al. (2020), and Liguori et al. (2023), we first perform a trial enumeration with a small tentative gap. For the CMTVRPTW and its two variants tested in this paper, our experiments suggest that setting the trial gap to be $\min\{1\% \times lb, 10\}$ works well, where lb is the current lower bound. For the CVRP and VRPTW tested in Section 7.5, we used $\min\{0.2\% \times lb, 40\}$ for standard instances and $\min\{0.2\% \times lb, 20\}$ for long instances. In practice, this value can be straightforwardly determined through several trials aimed at creating a pool of moderate size (containing 50,000 to 500,000 columns) in most cases. Subsequently, DeLuxing is applied to remove unnecessary columns from this enumerated trial pool. Finally, we solve an IP using the columns remaining in the pool. In case the pool still contains more than $\widehat{\beta}$ columns, we can keep the smallest $\widehat{\beta}$ ones based on their reduced costs.

This simple heuristic has proven highly effective. One of the main factors contributing to its success is that some columns essential for constructing high-quality feasible solutions might be absent in the current RMP but can be generated through the trial enumeration. DeLuxing plays a crucial role in this heuristic, as the trial enumeration can still produce a large number of columns. Nonetheless, a direct screening based solely on reduced costs, as described in the previous paragraph, performs badly. DeLuxing can often reduce the size of the column pool to be much smaller than $\widehat{\beta}$ while ensuring that necessary columns are retained in the pool.

7. Numerical Results

In this section, we present an extensive numerical study comprising five sets of experiments. The first set aims to show the effectiveness of the key components of DeLuxing in removing columns. In the second set, we individually evaluate the effectiveness of DeLuxing and each extension introduced in Section 6 using CMTVRPTW instances. The third set compares our default method (with DeLuxing and the three extensions enabled) with the state-of-the-art algorithms on the CMTVRPTW and its two important variants, CMTVRPTW-LT and CMTVRPTW-R. We exclude the other two variants, the CMTVRPTW *with limited trip duration* (CMTVRPTW-LD) and the *drone routing problem* (DRP) considered in Yang (2023) as the difficulty of solving them does not stem from generating excessive variables in the solution process. In fact, the number of routes generated in solving the CMTVRPTW-LD and DRP instances is relatively small (mostly several thousand for the CMTVRPTW-LD and tens of thousands for the DRP) according to tables EC.8 and EC.10 in Yang (2023). The fourth set of experiments seeks to further demonstrate the potential of the proposed approach by solving significantly larger multitrip instances, with sizes twice as large as the largest ones currently documented in the literature. The last set of experiments is dedicated to showing that the proposed DeLuxing and the primal heuristic also substantially accelerate one of the world's leading solvers, RouteOpt (You et al. 2023), for solving the CVRP and VRPTW instances. Note that RouteOpt incorporates the bucket arc elimination (Sadykov et al. 2021), a sophisticated type of arc-flow fixing. Throughout the solution process, RouteOpt also applies standard RCF.

For the CMTVRPTW, we consider two data sets, totaling 171 instances. The first set comprises 81 instances described in section 7.4.1 of Yang (2023), which are derived from the 27 type 2 Solomon instances. For each instance, we consider three cases: the first 70, the first 80, and all 100 customers. The second set consists of 90 large instances derived from the 30 instances (C2, R2, and RC2) in the G02 group (see Homberger and Gehring 2005). For each instance, we use the first 140, the first 170, and all 200 customers. The numbers of vehicles are set to 6, 7, 8, 12, 16, and 20 for instances with 70, 80, 100, 140, 170, and 200 customers, respectively, and the vehicle capacity is set to 100 for all the instances. For the CMTVRPTW-LT, we use the same 171 instances as the CMTVRPTW with the same parameters. The loading time of each customer is set to 20% of its service time following the procedure in Hernandez et al. (2016). For the CMTVRPTW-R, we use a total of 513 instances: 243 instances from section 7.4.4 of Yang (2023) generated from the 81 CMTVRPTW instances via the procedure in Cattaruzza et al. (2016a) with $\kappa \in \{0.25, 0.5, 0.75\}$ and an additional 270 instances generated from the 90 large CMTVRPTW instances using the same procedure. The number of vehicles and vehicle capacity are set to the same as those of the corresponding CMTVRPTW instances.

For classic VRPs, we employ the 200-node CVRP and 300-node VRPTW instances generated according to the procedure outlined in Uchoa et al. (2017). Specifically, the depot and customers are assigned integer coordinates within a $[0, 1000] \times [0, 1000]$ grid. The depot is placed at the center of the grid with coordinates (500, 500), and customer locations are randomly distributed across the grid. Customer demands are integer values uniformly drawn from [1, 100]. For the CVRP instances, the average route length, r, is uniformly selected from [6, 10], and the vehicle capacity is calculated by $Q = \lceil \frac{rD}{n} \rceil$, where D is the total demand of all customers and n is the number of customers. For the VRPTW instances, the vehicle capacity is set to 1,000, the service time is 1,000, and the maximum time resource H is 12,000. For 80% of the customers, their time windows are of the form $[0, b_i]$, where b_i is an integer uniformly selected from $[0, \lfloor 0.8H \rfloor]$. The remaining 20% of the customers have time windows $[a_i, b_i]$, with a_i and b_i, respectively, uniformly selected from $[\lfloor 0.2H \rfloor, \lfloor 0.8H \rfloor]$ and $[a_i, \min\{H, \lfloor a_i + 0.25H \rfloor\}]$. These 200 CVRP and 200 VRPTW instances feature branch-and-bound trees with

hundreds of nodes, and thus are generally challenging to solve. Meanwhile, the route length is moderate, making the pricing not overly difficult when many non-robust cuts, for example, limited memory rank-1 cuts (Pecin et al. 2017b), are present. Instances from the well-known CVRPLIB (Lima et al. 2014) are not ideal for demonstrating the advantages of our proposed methods because most of them are either too easy (resulting in trees with several nodes or even being solved at the root) or too challenging (unsolvable within a day) for cutting-edge solvers.

All experiments are conducted on a workstation running Ubuntu 20.04 equipped with an Intel® Core™ i9-12900K CPU @ 3.90 GHz and 128 GB of RAM. The code is implemented in C++ language and compiled by g++ 9.4.0. Gurobi 9.1.1 is used as an LP and IP solver. All LPs are solved in the single-thread mode, and eight threads are used to solve all IPs (MIPs). As a benchmark for our last set of experiments, the VRP-Solver is run on the same machine in the same mode (one thread for LPs and eight threads for IPs/MIPs) using CPLEX 22.1.0. The *k*-means++ method is run parallelly with all available threads. The time limit for each multitrip instance is set to three hours, and no limit is set for the CVRP and VRPTW instances. The source code and data needed to reproduce the results in this paper can be found in the e-companion.

7.1. Effectiveness of the Key Components of DeLuxing

In this section, we aim to demonstrate the effectiveness of the key components of DeLuxing. The **Benchmark** is Algorithm 1 with $p = 50$, $\beta_1 = 500,000$, and $\beta_2 = 50$. We consider the following four variants, where the parameters p, β_1, and β_2 are set to the same values as Benchmark unless otherwise specified.

1. **FullForm:** This variant modifies Substep 1 of Algorithm 2 to solve the full formulation $\widehat{\mathrm{F}}(\underline{\mathcal{R}}, \tilde{\mathcal{J}})$ instead of our formulation $\widehat{\mathrm{F}}(\mathcal{R}_1, \tilde{\mathcal{J}})$ enabled by the proposed Propositions 1 and 2.
2. **ClustByNorm:** This variant sets β_1 to zero and thus always applies the proposed ClustByNorm heuristic for the initial clustering.
3. **Random:** This variant randomly partitions $\underline{\mathcal{R}}$ into p clusters of equal size.
4. **NoDeepSearch:** This variant sets β_2 to $+\infty$ to skip the proposed deep search.

We compare the performance of these five methods on the column pools enumerated in the process of solving the CMTVRPTW instances. The advantages of the new formulation enabled by Propositions 1 and 2 can be demonstrated through a comparison of Benchmark and FullForm. The effectiveness of the straightforward heuristic approach, ClustByNorm, will be shown by comparing it with Benchmark and Random. Lastly, the benefits of the proposed deep search can be observed by comparing Benchmark with NoDeepSearch. The following information is included: the number of customers n, the average percentage of columns removed, and the average computing time (in seconds; CPU). Each average value is taken over all instances of the same size.

Figure 2 illustrates the performance comparison among different variants. Benchmark outperforms all other variants except FullForm, in terms of the percentage of columns removed. This superiority becomes more pronounced when dealing with larger instances, where the number of customers is higher and the challenges are greater. It is important to note that even a 1% increase in the removal percentage translates to thousands of additional variables being eliminated, considering the average pool size of over 100,000 columns. Such extra reductions in variables drastically impact the overall algorithmic performance. Although Benchmark may not remove as many columns as FullForm, it manages to reduce the computational time to less than 1/10 that of FullForm for instances with 140 or fewer customers. Such CPU reductions are crucial for the success of DeLuxing and highlight the significance of the new formulation. It is worth mentioning that FullForm hits the three-hour time limit for most 170- and 200-customer R and RC instances, which is the reason why the CPU differences between Benchmark and FullForm are less significant.

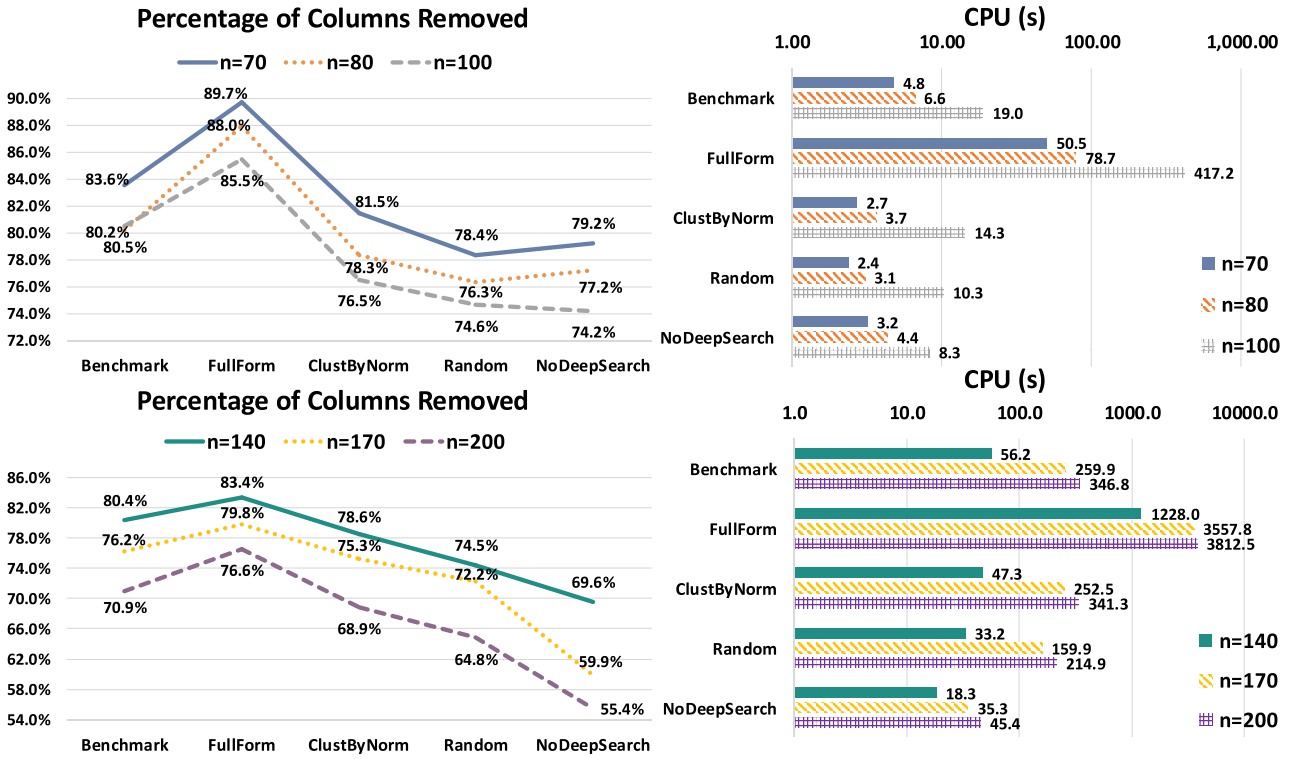
Comparing ClustByNorm with Random and Benchmark, we conclude that ClustByNorm is effective, exhibiting much better performance than random initial clustering, albeit slightly inferior to Benchmark. Furthermore, the computational overhead associated with ClustByNorm is smaller than Benchmark. The importance of the proposed deep search is evident when comparing Benchmark with NoDeepSearch. Notably, for large instances, that is, those with 140 or more customers, the proposed deep search substantially increases the percentage of columns removed.

7.1.1. Sensitivity Analysis.

In this section, we aim to justify our selection of values for the three parameters p, β_1, and β_2 and illustrate their sensitivity. We conduct tests with varying values: $p \in \{10, 20, 50, 100\}$, $\beta_1 \in \{100,000;\ 200,000;\ 500,000;\ 1,000,000\}$, and $\beta_2 \in \{10, 50, 100\}$, resulting in a total of 48 combinations. For each combination, we run an experiment on the same 30 column pools from the 30 instances with 170 customers. We present the average percentage of columns removed and the average CPU in seconds.

As shown in Figure 3, our chosen values of $p = 50$, $\beta_1 = 500,000$, and $\beta_2 = 50$ achieve a 76.2% column reduction in 259.9 seconds, striking a suitable balance between the percentage of columns removed and the computational time. Moreover, the parameters are not overly sensitive to variations. In particular, multiple combinations, such as (1) $p = 50$, $\beta_1 = 100,000$, $\beta_2 = 50$; (2) $p = 50$, $\beta_1 = 200,000$, $\beta_2 = 50$; (3) $p = 100$, $\beta_1 = 200,000$, $\beta_2 = 100$;

Figure 2. (Color online) Comparison of Percentage of Columns Removed and CPU in Seconds for the CMTVRPTW



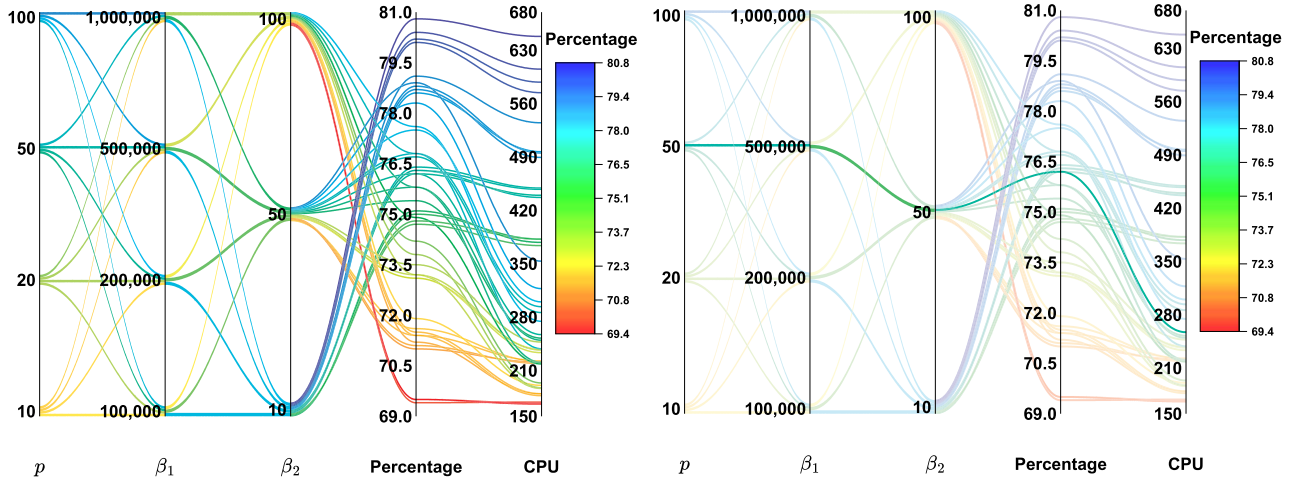
Notes. Each number is the average value taken over instances of the same size. The x-axis is in logarithmic scale.

and (4) $p = 100$, $\beta_1 = 500,000$, $\beta_2 = 100$, achieve a comparable reduction of around 76% within the range of 220 to 260 seconds. Detailed results are summarized in Table EC.2 in Section EC.1 of the e-companion.

7.2. Effectiveness of DeLuxing and Three Extensions

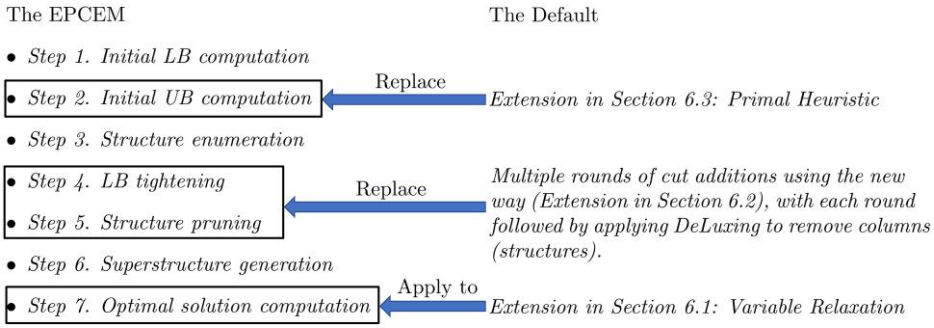
We demonstrate the isolated effectiveness of DeLuxing and the three inspired extensions described in Section 6. Our baseline method, denoted by Default, is a modified implementation of the EPCEM from Yang (2023) with DeLuxing and the three extensions incorporated as follows (see Figure 4 for an illustration). The extension in Section 6.3 (our new primal heuristic) is applied in Step 2 in place of the standard primal heuristic to compute a valid UB. In addition, we replace Steps 4 and 5 of the EPCEM with Step 4′, which performs multiple rounds of alternating cut addition and column removal by

Figure 3. (Color online) Comparison of Average Percentage of Columns Removed and Average CPU in Seconds for the 170-Customer Instances



Note. The right panel highlights the choice of $p = 50$, $\beta_1 = 500,000$, and $\beta_2 = 50$.

Figure 4. (Color online) Illustration of Development of Default from the EPCEM in Yang (2023)



Note. LB, lower bound.

DeLuxing. In each round, we add cuts using the extension in Section 6.2 (the new way of cut addition), followed by DeLuxing to remove columns. Finally, the extension in Section 6.1 (the variable relaxation) is applied to relax the IP in Step 7 and calculate the optimal solution by solving the resulting MIP.

We disable each component separately on top of Default each time, and the resulting settings are denoted by NoDeluxing, NoVarRelax, OldCutAdd, and OldPrimalHeu (applying the traditional primal heuristic used in the EPCEM to compute a UB), respectively. We report the number of customers (n), the number of instances of this size (#Inst), the number of instances solved to optimality (Solved), and the average optimality gap $\frac{ub-lb}{ub} \times 100\%$ at termination (Gap%). The Gap is averaged over instances that cannot be solved optimally within the time limit.

According to Table 1 and Figure 5, Default solves significantly more instances than NoDeLuxing and OldPrimalHeu while being 48%, 21%, 161%, 115%, 18%, and 26% faster than NoDeLuxing, and 106%, 206%, 813%, 249%, 38%, and 67% faster than OldPrimalHeu, respectively, for instances of sizes 70 to 200. These results confirm the high effectiveness of DeLuxing in accelerating the algorithm and its essential contribution to solving challenging instances. Furthermore, the adoption of the primal heuristic, in which DeLuxing plays a pivotal role, significantly enhances the algorithm's capability to solve large instances by providing tight upper bounds at an early stage. Although the variable relaxation and the new approach for cutting plane addition may have limited effectiveness for small-sized instances, they prove to be valuable in achieving optimality faster for larger instances. In particular, Default outperforms NoVarRelax by solving two more instances and is 14% faster for 140-customer instances. Although Default and OldCutAdd solve the same total number of instances, Default surpasses OldCutAdd by being 25% faster for instances of size 140.

7.3. Comparison with State-of-the-Art Algorithms

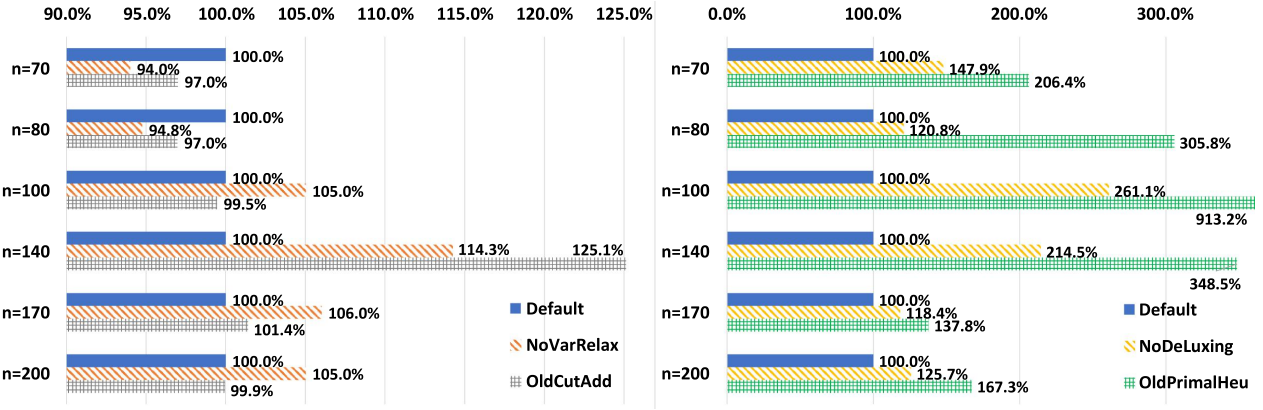
In this section, we compare our Default with three state-of-the-art algorithms: Yang (2023), Roboredo et al. (2023), and Zhang (2022). It is worth mentioning that our hardware is better than others. To ensure a fair comparison, we scale the computational times of the other three methods based on their CPU frequencies. More precisely, the CPU frequencies reported in Yang (2023), Roboredo et al. (2023), and Zhang (2022) are 3.7 GHz, 3.6 GHz, and 2.9 GHz, respectively, which necessitates dividing their reported times by a factor of 1.05, 1.08, and 1.34.

In Tables 2–5, we report the values of n, #Inst, Solved, Gap, and the computational time in seconds (CPU). Detailed results for each instance are reported in Tables EC.3 to EC.5 in Section EC.2 of the e-companion. The CPU values presented in Tables 2–5 are averaged over **all** instances of the same size and are scaled values for the three benchmark methods. If an instance cannot be solved to optimality within the three-hour time limit, its

Table 1. Summary of the Results for the CMTVRPTW

		Default		NoDeLuxing		NoVarRelax		OldCutAdd		OldPrimalHeu	
n	#Inst	Solved	Gap%	Solved	Gap%	Solved	Gap%	Solved	Gap%	Solved	Gap%
70	27	27	0.0	27	0.0	27	0.0	27	0.0	27	0.0
80	27	27	0.0	27	0.0	27	0.0	27	0.0	27	0.0
100	27	27	0.0	27	0.0	27	0.0	27	0.0	23	2.6
140	30	29	0.1	27	0.4	30	0.0	30	0.0	21	2.2
170	30	22	0.5	18	0.4	21	0.5	22	0.5	15	1.6
200	30	22	0.4	17	0.5	20	0.3	21	0.4	12	1.4

Figure 5. (Color online) Comparison on CPU



Notes. Each number represents the ratio of the average CPU time compared with that of the Default method, where each average value is taken over instances of the same size. The CPU values are recorded as 10,800 for instances that cannot be solved to optimality within the three-hour time limit.

CPU value is recorded as 10,800 even though it may be terminated early because of insufficient memory. Note that Zhang (2022) only reported results for instances of sizes 80 and 100. Moreover, Roboredo et al. (2023) did not experiment with the two variants considered in this paper and did not report the optimality gap at termination. Roboredo et al. (2023) reported results for two settings, that is, with or without initial UB. For consistency with other methods, we use the setting without UB in Table 2.

7.3.1. Comparison on the CMTVRPTW. As shown in Table 2, our method can solve all 81 CMTVRPTW instances optimally while being, on average, more than 10 and 7 times, respectively, as fast as Zhang (2022) for instances of sizes 70 and 100. In contrast, Yang (2023) can only solve 65 out of the 81 instances to optimality and is more than 20 times slower than our method. For a fair comparison with Roboredo et al. (2023), we conducted two additional sets of experiments: one with the optimal value as an initial upper bound and the other without the upper bound. Both experiments were run using a single thread for solving all LPs and IPs, and the results are summarized in Table 3. In both cases, our method can still solve all instances optimally whereas Roboredo et al. (2023) solve 51 and 67 instances, respectively, without/with an upper bound.

7.3.2. Comparison on the CMTVRPTW-LT. According to Table 4, our method consistently outperforms all the benchmark algorithms substantially on the CMTVRPTW-LT. Specifically, it can solve all 81 instances optimally, with computational speeds more than 10 times for 70-customer instances and over 5 times for 100-customer instances as fast as those of Zhang (2022). In contrast, Yang (2023) only solves 64 of the 81 instances optimally, and it once again takes over 20 times more time than our method.

7.3.3. Comparison on the CMTVRPTW-R. Table 5 summarizes the results for 243 CMTVRPTW-R instances. Our method can solve all but one instance optimally and achieves an optimality gap of 0.3% for the only unsolved instance. In terms of computational speed, our method is, once again, significantly faster than Zhang (2022) and Yang (2023).

7.4. Computational Results for Large Multitrip Instances

In this section, we test Default on significantly larger instances with sizes twice as large as the largest ones currently documented in the literature, which are exponentially more difficult to solve. For the CMTVRPTW and CMTVRPTW-LT, the CPU of an instance whose optimality cannot be proved within the three-hour time limit is again counted as 10,800,

Table 2. Comparison on the CMTVRPTW Using Eight Threads

		This paper (Default)			Yang (2023)			Zhang (2022)		
n	#Inst	Solved	Gap%	CPU	Solved	Gap%	CPU	Solved	Gap%	CPU
70	27	27	0.0	26.7	27	0.0	1,230.3	27	0.0	343.9
80	27	27	0.0	49.6	24	1.0	2,197.5	—	—	—
100	27	27	0.0	240.5	14	1.3	7,122.5	27	0.0	1,686.4

Table 3. Comparison on the CMTVRPTW with and Without UB Using One Thread

		Default (one thread) without UB			Default (one thread) with UB			Roboredo et al. (2023) without UB			Roboredo et al. (2023) with UB		
n	#Inst	Solved	Gap%	CPU	Solved	Gap%	CPU	Solved	Gap%	CPU	Solved	Gap%	CPU
70	27	27	0.0	39.0	27	0.0	20.1	22	—	3,443.6	25	—	2,511.3
80	27	27	0.0	83.1	27	0.0	43.5	18	—	4,446.9	25	—	2,261.7
100	27	27	0.0	570.0	27	0.0	589.2	11	—	6,357.9	17	—	5,380.1

and the values of Gap and CPU are averaged over **all** instances. However, for some CMTVRPTW-R instances, no feasible solution can be found at termination. Such instances (3, 2, and 9 instances of sizes 140, 170, and 200, respectively; 14 in total) are excluded from the computation of Gap and CPU values. Table 6 summarizes the results, and more details can be found in Tables EC.3 to EC.5 in Section EC.2 of the e-companion.

According to Table 6, all but one CMTVRPTW instance of size 140 can be solved within three hours, and the optimality of the only unsolved one can be proved within five hours. Among the 60 CMTVRPTW instances of sizes 170 and 200, 44 instances can be solved. The average gaps for the unsolved instances are approximately 0.5% and 0.4%, respectively. Our method achieves very similar results for the CMTVRPTW-LT: it solves all but two instances of size 140. In addition, 70% of 170- and 200-customer instances can be solved, and the average gaps of the unsolved ones are 0.5%. For the CMTVRPTW-R, around 93%, 68%, and 52% of instances of sizes 140, 170, and 200 can be solved, respectively. The average gaps of the unsolved instances are all below 1.5%. The optimality of all solved instances can be proved, on average, in less than 16 minutes. These results clearly demonstrate that the Default brings our capabilities of solving CMTVRPTW, CMTVRPTW-LT, and CMTVRPTW-R to an entirely new level.

7.5. Computational Results for CVRP and VRPTW Instances

In this section, we focus on demonstrating the effectiveness of DeLuxing and the new primal heuristic from Section 6.3 for solving two classic types of VRPs: CVRP and VRPTW. For each problem class, we conduct six sets of experiments: the basic VRP-Solver, basic RouteOpt (RouteOpt0), RouteOpt with DeLuxing (RouteOpt1), RouteOpt with the traditional (old) primal heuristic (RouteOpt2), RouteOpt with our new primal heuristic (RouteOpt3), and RouteOpt with both DeLuxing and the new primal heuristic (RouteOpt4). The initial upper bound is set to $ub^0 = (1 + 0.1\%) \cdot v^*$ so that all instances can be solved to optimality while ensuring the enumeration does not often succeed at the root, where v^* is the optimal value.

It is important to note that branching proves ineffective in solving the multitrip instances in previous sections and, thus, is not performed. In such scenarios, we apply DeLuxing aggressively to make the final IP as small as possible because the primary challenge lies in solving this IP. However, this is not the case anymore when solving the CVRP and VRPTW. Branching is indispensable, leading to the generation of many nodes and, thus, IPs to solve. Applying DeLuxing immediately after a successful enumeration at each BBN becomes inefficient. Instead, branching can initially downsize an enumerated pool rapidly. We further observe that in a later stage, branching does not reduce the pool effectively and ends up creating too many nodes.

The goal of DeLuxing, in this case, shifts to lessening dependence on branching. It closes each node early, thereby reducing the BB tree size. Specifically, DeLuxing is applied moderately at each node once the pool size falls below a set threshold, denoted as $\overline{N}$. The reduced IP can then be solved efficiently by Gurobi, and thus, the node is closed without further branching. Through experimentation, we found that setting $p = 20$, $\beta_1 = 1,000$, $\beta_2 = 1,000$, and $\overline{N} = 50,000$ typically halves the pool size within 10 seconds across most instances. Consequently, these parameters are adopted across all experiments in this section.

7.5.1. Effectiveness of DeLuxing. Table 7 summarizes the average time in seconds (CPU) and the tree size measured by the number of BBNs (Node). For detailed

Table 4. Comparison on the CMTVRPTW-LT

		This paper (Default)			Yang (2023)			Zhang (2022)		
n	#Inst	Solved	Gap%	CPU	Solved	Gap%	CPU	Solved	Gap%	CPU
70	27	27	0.0	27.4	27	0.0	1,497.0	27	0.0	329.7
80	27	27	0.0	48.4	24	1.1	2,558.2	—	—	—
100	27	27	0.0	362.3	13	1.2	7,201.4	27	0.0	1,868.4

Table 5. Comparison on the CMTVRPTW-R

			This paper (Default)			Yang (2023)			Zhang (2022)		
n	κ	#Inst	Solved	Gap%	CPU	Solved	Gap%	CPU	Solved	Gap%	CPU
70	0.25	27	27	0.0	23.2	27	0.0	251.9	27	0.0	190.8
70	0.50	27	27	0.0	13.8	27	0.0	190.2	27	0.0	164.8
70	0.75	27	27	0.0	24.9	26	0.6	490.6	27	0.0	156.5
80	0.25	27	27	0.0	17.9	27	0.0	1,055.8	—	—	—
80	0.50	27	27	0.0	26.4	27	0.0	433.7	—	—	—
80	0.75	27	27	0.0	50.0	27	0.0	542.7	—	—	—
100	0.25	27	27	0.0	196.8	23	2.0	3,534.8	27	0.0	771.6
100	0.50	27	26	0.3	583.5	23	1.4	3,140.9	27	0.0	743.4
100	0.75	27	27	0.0	182.7	21	1.6	3,288.0	27	0.0	536.5

results, readers are referred to Tables EC.6 and EC.7 in Section EC.3 of the e-companion. Without DeLuxing, RouteOpt is already, on average, about 90% faster than the VRP-Solver for the CVRP instances and 42% faster for the VRPTW instances. DeLuxing further speeds up it by 21% and 31%, respectively, resulting in a nearly 2.3× speed for the CVRP and a 1.9× speed for the VRTPW compared with the VRP-Solver. This acceleration is primarily attributed to DeLuxing consistently reducing the tree size by more than two-thirds. This improvement is even more pronounced for difficult instances: the 76 CVRP and 100 VRPTW instances that take more than an hour to solve using the VRP-Solver. RouteOpt1 achieves over 2.5 times and nearly 2 times the speed of the VRP-Solver, respectively, for these CVRP and VRPTW instances.

7.5.2. Effectiveness of Our Primal Heuristic. We observe in our experiments that the standard primal heuristic yields a solution of better value than the given initial upper bound for only 43 out of the 200 CVRP instances and 64 out of the 200 VRPTW instances. According to Table 7, its application in RouteOpt results in little difference: 1% speedup for the CVRP instances and 1% slowdown for the VRPTW instances. By comparing RouteOpt0 and RouteOpt3, we conclude that our proposed primal heuristic accelerates RouteOpt by around 31% and 41%, respectively, substantially outperforming the standard primal heuristic.

7.5.3. Effectiveness of DeLuxing and Our Primal Heuristic Combined. Comparing RouteOpt1 and RouteOpt4 in Table 7, we can conclude that even when DeLuxing is enabled, our primal heuristic can still accelerate RouteOpt by around 20% and 31%, respectively. When DeLuxing and our primal heuristic are both enabled, they reduce the tree size by more than 80%, making RouteOpt 45% and 71% faster, respectively. Overall, the speed is about 175% and 143%, respectively, faster than the VRP-Solver for solving the CVRP and VRPTW instances.

7.5.4. Results for Long CVRP Instances. The 400 CVRP and VRPTW instances tested have moderate route lengths: the average number of customers per route is 7.1 (with a maximum of 15) for the CVRP and 7.9 (with a maximum of 10) for the VRPTW. For these instances, enumeration typically succeeds relatively early in the solution process, with the first successful enumeration occurring at around 14.3% and 4.7% of the whole process for the CVRP and VRPTW, respectively, and the last success happening at 87.1% and 84.7% of the duration, respectively. Despite early enumeration, the branching tree remains large, with average tree sizes of 171.5 and 449.9, respectively, when solved under setting RouteOpt0 (see Table 7).

In instances with long routes, successful enumeration tends to occur much later in the process. Once successful, branching can rapidly close a node, resulting in much smaller BB trees. In such cases, DeLuxing is expected to be less effective. To verify this intuition, we conducted additional experiments on long CVRP instances. To this end, we increased the capacity of the 200 CVRP instances by 80%, on average lengthening the routes to 15.6 (with a maximum of 23). We obtained optimal solutions for 65 of these instances and used

Table 6. Computational Results for Large Instances

	CMTVRPTW				CMTVRPTW-LT				CMTVRPTW-R			
n	#Inst	Solved	Gap%	CPU	#Inst	Solved	Gap%	CPU	#Inst	Solved	Gap%	CPU
140	30	29	0.1	1,254.0	30	28	0.2	1,372.5	90	84	1.4	365.9
170	30	22	0.5	4,210.7	30	21	0.5	4,048.4	90	61	1.1	722.0
200	30	22	0.4	4,306.3	30	21	0.5	4,649.6	90	47	1.3	930.0

Table 7. Computational Results for the CVRP and VRPTW Instances

				VRP-Solver		RouteOpt0		RouteOpt1		RouteOpt2		RouteOpt3		RouteOpt4	
	n	#Inst	Solved	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node	CPU	Node
CVRP	200	200	200	4,495.5	158.2	2,369.9	171.5	1,959.8	55.0	2,342.5	166.3	1,814.0	94.7	1,635.5	30.4
VRPTW	300	200	200	5,574.4	152.3	3,915.1	449.9	2,993.9	141.3	3,949.5	450.5	2,779.1	290.4	2,293.8	88.3

them for the following analysis. As expected, the first successful enumeration occurs much later, at around 27.8%, and the last happens at 99.8% of the process. Additionally, the tree size is significantly smaller, with an average of 63.7. In this scenario, our experiments indicate that DeLuxing is no longer effective, as nodes are closed rapidly by branching following successful enumeration. Nonetheless, our primal heuristic still accelerates RouteOpt by more than 25%. Detailed results are included in Table EC.8 in Section EC.3 of the e-companion.

8. Concluding Remarks

We propose a highly effective variable fixing strategy, called DeLuxing, that employs a novel deep search method for identifying promising dual solutions. Based on theoretical results, it solves a novel LP formulation with only a small subset of the enumerated variables in each iteration. DeLuxing can remove more than 75% variables in most cases, achieving a direct acceleration of over 50%. Enhanced by the additional three extensions inspired by DeLuxing, our method can be more than 7 times on average and up to more than 20 times as fast as the best-performing exact method in the literature. In particular, our method can solve all but one CMTVRPTW instance with 140 customers in three hours and prove optimality for the remaining one in five hours, which doubles the size of previously completely solvable instances. Significant performance improvement is also achieved for the two important variants (the CMTVRPTW-LT and CMTVRPTW-R) and two classic VRPs (the CVRP and VRPTW). It is worth mentioning that the enumeration of all necessary columns early in the solution process is crucial to the success of DeLuxing. When the routes are long, DeLuxing is less effective because enumeration may not succeed until a late stage. Nonetheless, the new primal inspired by DeLuxing performs consistently well across all tested instances in this paper, even when full enumeration only succeeds late.

Currently, in the subroutine of DeLuxing (Algorithm 2), we employ the dual simplex method or interior point method without crossover to solve the LP formulation $\widehat{\mathrm{F}}(\mathcal{R}_1, \tilde{\mathcal{J}})$ and obtain each time one optimal dual solution for determining the set of removable columns $\mathcal{D}$. In future research, it would be beneficial to explore the possibility of recording all feasible dual solutions encountered during the solution process and utilizing them to compute LUs for further variable fixing. Another potential research direction is to investigate the similarities among columns in an artificial cluster $\mathcal{D} \setminus \tilde{\mathcal{J}}$ and develop even more effective approaches to bundle columns for computing qualified dual solutions. In addition, extending the basic principle underpinning DeLuxing to other contexts such as the pricing algorithm and branching variable selection can potentially lead to extra acceleration. Finally, establishing theoretical guarantees, in a probabilistic sense, regarding the performance of DeLuxing under potentially mild assumptions can also be an interesting research direction.

Acknowledgments

The author extends sincere thanks to the area editor, associate editor, and the three anonymous referees, whose insightful feedback has greatly improved this paper.

References

Achterberg T (2018) Exploiting degeneracy in MIP. Presentation, Aussois 22nd Combinatorial Optimization Workshop, CNRS Centre Paul Langevin, Aussois, France, http://www.iasi.cnr.it/aussois/web/uploads/2018/slides/achterbergt.pdf.

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 AI OR Techniques Constraint Programming Combin. Optim. Problems 5th Internat. Conf. CPAIOR 2008* (Springer, Berlin), 6–20.

Aloise D, Deshpande A, Hansen P, Popat P (2009) NP-hardness of Euclidean sum-of-squares clustering. *Machine Learn.* 75:245–248.

Amaldi E, Kann V (1995) The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoret. Comput. Sci.* 147(1–2):181–210.

Arthur D, Vassilvitskii S (2007) K-means++ the advantages of careful seeding. Gabow H, ed. *Proc. 18th Annual ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 1027–1035.

Bacchus F, Hyttinen A, Järvisalo M, Saikko P (2017) Reduced cost fixing in MaxSAT. Beck JC, ed. *Principles Practice Constraint Programming 23rd Internat. Conf. CP 2017* (Springer, Berlin), 641–651.

Bajgiran OS, Cire AA, Rousseau LM (2017) A first look at picking dual variables for maximizing reduced cost fixing. Salvagnin D, Lombardi M, eds. *Integration AI OR Techniques Constraint Programming 14th Internat. Conf. CPAIOR 2017* (Springer, Berlin), 221–228.

Balas E, Carrera MC (1996) A dynamic subgradient-based branch-and-bound procedure for set covering. *Oper. Res.* 44(6):875–890.

Balas E, Saltzman MJ (1991) An algorithm for the three-index assignment problem. *Oper. Res.* 39(1):150–161.

Baldacci R, Bartolini E, Mingozzi A (2011a) An exact algorithm for the pickup and delivery problem with time windows. *Oper. Res.* 59(2):414–426.

Baldacci R, Christofides N, Mingozzi A (2008) An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Math. Programming* 115(2):351–385.

Baldacci R, Hadjiconstantinou E, Mingozzi A (2004) An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Oper. Res.* 52(5):723–738.

Baldacci R, Mingozzi A, Calvo RW (2011d) An exact method for the capacitated location-routing problem. *Oper. Res.* 59(5):1284–1296.

Baldacci R, Mingozzi A, Roberti R (2011c) New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.* 59(5):1269–1283.

Baldacci R, Mingozzi A, Roberti R (2012) New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS J. Comput.* 24(3):356–371.

Baldacci R, Bartolini E, Mingozzi A, Valletta A (2011b) An exact algorithm for the period routing problem. *Oper. Res.* 59(1): 228–241.

Baldacci R, Mingozzi A, Roberti R, Calvo RW (2013) An exact algorithm for the two-echelon capacitated vehicle routing problem. *Oper. Res.* 61(2):298–314.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MW, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Bixby RE (2002) Solving real-world linear programs: A decade and more of progress. *Oper. Res.* 50(1):3–15.

Bixby ER, Fenelon M, Gu Z, Rothberg E, Wunderling R (2000) MIP: Theory and practice—closing the gap. Powell MJD, Scholtes S, eds. *System Model. Optim. CMSO 1999* (Springer, Boston), 19–49.

Bradley PS, Fayyad UM (1998) Refining initial points for k-means clustering. Shavlik JW, ed. *Proc. 15th Internat. Conf. Machine Learn.* (Morgan Kaufmann Publishers Inc., San Francisco), 91–99.

Breugem T, Dollevoet T, Huisman D (2022) Is equality always desirable? Analyzing the trade-off between fairness and attractiveness in crew rostering. *Management Sci.* 68(4):2619–2641.

Cappanera P, Gallo G (2004) A multicommodity flow approach to the crew rostering problem. *Oper. Res.* 52(4):583–596.

Cattaruzza D, Absi N, Feillet D (2016a) The multi-trip vehicle routing problem with time windows and release dates. *Transportation Sci.* 50(2):676–693.

Cattaruzza D, Absi N, Feillet D (2016b) Vehicle routing problems with multiple trips. *4OR* 14(3):223–259.

Celebi ME, Kingravi HA, Vela PA (2013) A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems Appl.* 40(1):200–210.

Cheng C, Adulyasak Y, Rousseau LM (2020) Drone routing with energy function: Formulation and exact algorithm. *Transportation Res. Part B Methodological* 139:364–387.

Contardo C, Martinelli R (2014) A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optim.* 12:129–146.

Costa L, Contardo C, Desaulniers G (2019) Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Sci.* 53(4):946–985.

Crainic TG, Maggioni F, Perboli G, Rei W (2018) Reduced cost-based variable fixing in two-stage stochastic programming. *Ann. Oper. Res.* 1–37.

Crowder H, Johnson EL, Padberg M (1983) Solving large-scale zero-one linear programming problems. *Oper. Res.* 31(5):803–834.

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.

Dantzig G, Fulkerson R, Johnson S (1954) Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Amer.* 2(4):393–410.

de Lima VL, Iori M, Miyazawa FK (2023) Exact solution of network flow models with strong relaxations. *Math. Programming* 197(2): 813–846.

Desaulniers G, Gschwind T, Irnich S (2020) Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. *Transportation Sci.* 54(5):1170–1188.

Desaulniers G, Rakke JG, Coelho LC (2016b) A branch-price-and-cut algorithm for the inventory-routing problem. *Transportation Sci.* 50(3):1060–1076.

Desaulniers G, Errico F, Irnich S, Schneider M (2016a) Exact algorithms for electric vehicle-routing problems with time windows. *Oper. Res.* 64(6):1388–1405.

Engineer FG, Furman KC, Nemhauser GL, Savelsbergh MW, Song JH (2012) A branch-price-and-cut algorithm for single-product maritime inventory routing. *Oper. Res.* 60(1):106–122.

Ford LR, Fulkerson DR (1958) A suggested computation for maximal multi-commodity network flows. *Management Sci.* 5(1):97–101.

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

Gurobi Optimization, LLC (2023) *Gurobi Optimizer Reference Manual* (Gurobi Optimization, LLC, Beaverton, OR).

Hernandez F, Feillet D, Giroudeau R, Naud O (2014) A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4OR* 12(3):235–259.

Hernandez F, Feillet D, Giroudeau R, Naud O (2016) Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows. *Eur. J. Oper. Res.* 249(2):551–559.

Holmberg K, Yuan D (2000) A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Oper. Res.* 48(3):461–481.

Homberger J, Gehring H (2005) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *Eur. J. Oper. Res.* 162(1):220–238.

Hooker JN, Ottosson G, Thorsteinsson ES, Kim HJ (1999) On integrating constraint propagation and linear programming for combinatorial optimization. *Proc. Sixteenth National Conf. Artificial Intelligence (Orlando, Florida)*, 136–141.

Irnich S, Desaulniers G, Desrosiers J, Hadjar A (2010) Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS J. Comput.* 22(2):297–313.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Johnson EL, Kostreva MM, Suhl UH (1985) Solving 0-1 integer programming problems arising from large scale planning models. *Oper. Res.* 33(4):803–819.

Kohl N, Desrosiers J, Madsen OB, Solomon MM, Soumis F (1999) 2-path cuts for the vehicle routing problem with time windows. *Transportation Sci.* 33(1):101–116.

Land AH, Doig AG (2010) An automatic method for solving discrete programming problems. Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, eds. *50 Years of Integer Programming 1958–2008* (Springer, Berlin), 105–132.

Laporte G, Nobert Y (1983) A branch and bound algorithm for the capacitated vehicle routing problem. *Oper. Res. Spektrum* 5:77–85.

Liguori PH, Mahjoub AR, Marques G, Sadykov R, Uchoa E (2023) Nonrobust strong knapsack cuts for capacitated location routing and related problems. *Oper. Res.* 71(5):1577–1595.

Lima I, Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A, Oliveira D, Queiroga E (2014) CVRPLIB: Capacitated vehicle routing problem library. http://vrp.galgos.inf.puc-rio.br/index.php/en/.

Lloyd S (1982) Least squares quantization in PCM. *IEEE Trans. Inform. Theory* 28(2):129–137.

Lysgaard J, Letchford AN, Eglese RW (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Programming* 100:423–445.

Marques G, Sadykov R, Deschamps JC, Dupas R (2020) An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Comput. Oper. Res.* 114:104833.

Mingozzi A, Roberti R, Toth P (2013) An exact algorithm for the multitrip vehicle routing problem. *INFORMS J. Comput.* 25(2): 193–207.

Paradiso R, Roberti R, Laganá D, Dullaert W (2020) An exact solution framework for multitrip vehicle-routing problems with time windows. *Oper. Res.* 68(1):180–198.

Pecin D, Contardo C, Desaulniers G, Uchoa E (2017a) New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS J. Comput.* 29(3):489–502.

Pecin D, Pessoa A, Poggi M, Uchoa E (2017b) Improved branch-cut-and-price for capacitated vehicle routing. *Math. Programming Comput.* 9(1):61–100.

Pessoa A, Uchoa E, Poggi de Aragão M (2009) A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks* 54(4):167–177.

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

Pessoa A, Uchoa E, de Aragão MP, Rodrigues R (2010) Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Math. Programming Comput.* 2: 259–290.

Posta M, Ferland JA, Michelon P (2012) An exact method with variable fixing for solving the generalized assignment problem. *Comput. Optim. Appl.* 52:629–644.

Quesnel F, Desaulniers G, Soumis F (2020) Improving air crew rostering by considering crew preferences in the crew pairing problem. *Transportation Sci.* 54(1):97–114.

Roberti R, Ruthmair M (2021) Exact methods for the traveling salesman problem with drone. *Transportation Sci.* 55(2): 315–335.

Roboredo M, Sadykov R, Uchoa E (2023) Solving vehicle routing problems with intermediate stops using VRPSolver models. *Networks* 81(3):399–416.

Sadykov R, Uchoa E, Pessoa A (2021) A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Sci.* 55(1):4–28.

Sellmann M (2004) Theoretical foundations of CP-based Lagrangian relaxation. Wallace M, ed. *Principles Practice Constraint Programming CP 2004* (Springer, Berlin), 634–647.

Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A (2017) New benchmark instances for the capacitated vehicle routing problem. *Eur. J. Oper. Res.* 257(3):845–858.

Wolsey LA, Nemhauser GL (1999) *Integer and Combinatorial Optimization* (John Wiley & Sons, New York).

Yang Y (2023) An exact price-cut-and-enumerate method for the capacitated multitrip vehicle routing problem with time windows. *Transportation Sci.* 57(1):230–251.

You Z, Yang Y, Wang X, Yin W (2023) Two-stage learning to branch in branch-price-and-cut algorithms for solving vehicle routing problems exactly. Preprint, submitted November 13, http://dx.doi.org/10.2139/ssrn.4630549.

Yunes T, Aron ID, Hooker JN (2010) An integrated solver for optimization problems. *Oper. Res.* 58(2):342–356.

Zhang S (2022) Solving the capacitated multi-trip vehicle routing problem with time windows. MPhil thesis, Hong Kong Polytechnic University, Hong Kong.

Yu Yang is an assistant professor of industrial and systems engineering at the University of Florida (UF) and an assistant director of the UF Center for Applied Optimization. His research interests are in combinatorial optimization, learning to optimize, large-scale optimization theory, and their applications in sustainable energy systems, supply chain management, and logistics. His team has developed RouteOpt, a leading open-source exact VRP solver.