Integer programming column generation: Accelerating branch-and-price using a novel pricing scheme for finding high-quality solutions in set covering, packing, and partitioning problems

Stephen J. Maher*

College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, United Kingdom

Elina Rönnberg†

Department of Mathematics, Linköping University, SE-581 83, Linköping, Sweden

Abstract

Large-neighbourhood search (LNS) heuristics are important mathematical programming techniques that search for primal feasible solutions by solving an auxiliary problem with a restricted feasible region. Extending such powerful generic LNS heuristics to the branch-and-price context is inherently challenging. The most prominent challenges arise from the fact that in branch-and-price algorithms, columns are generated with the sole aim to solve linear programming relaxations. Hence, the ability to form integer feasible solutions is not considered during the generation of columns. Without any changes to the standard pricing schemes, the potential of deploying generic LNS heuristics within a branch-and-price procedure is severely limited. This paper proposes a matheuristic, based on an LNS heuristic framework, where the novelty is a customised pricing scheme for generating columns to solve an auxiliary problem. The theoretical foundation for this pricing scheme is a set of optimality conditions for integer programs. From this foundation, a column generation

*s.j.maher@exeter.ac.uk

†elina.ronnberg@liu.se

strategy is developed for finding columns that are likely to be of use in high-quality primal feasible solutions for the original problem. The proposed matheuristic is implemented in the generic branch-price-and-cut solver GCG. On a broad test set comprising classical block diagonal structured instances and general instances from the MIPLIB 2017 Collection, the computational results show a significant improvement to the solving performance of GCG.
Key words: column generation, branch-and-price, matheuristic, large neighbourhood search

1 Introduction

Branch-and-price (Barnhart et al., 1998) is a mathematical programming technique for solving Mixed-Integer Programming (MIP) problems comprising a huge number of variables. When branch-and-price is applied to such problems, the variable set is not explicitly given but implicitly defined as the feasible solutions to a pricing problem. These variables—or columns as they are called interchangeably—are generated when solving Linear Programming (LP) relaxations of the MIP by column generation (Lübbecke and Desrosiers, 2005) at each node of a branch-and-bound tree. The problem formulation used in the branch-and-price context is often referred to as an extended formulation. This extended formulation is derived through a Dantzig-Wolfe reformulation of its compact counterpart, where the variables are explicitly given. An important characteristic of an extended formulation is that it is always at least as strong as the compact formulation. In particular, when the pricing problem does not have integrality property it is stronger, and in practice often much stronger. A price paid for this strength is the dramatic increase in the number of variables and the fact that a pricing problem needs to be solved to generate each variable.

A critical factor for the success of methods based on column generation is that only a small subset of the variables need to be generated to find an optimal solution. When designing a branch-and-price scheme, it is therefore important to avoid wasting computational time on generating columns that are of no use in high-quality solutions to the MIP. Note that when new columns are generated only to solve the LP relaxations in the branch-and-bound nodes, this is done without considering if they are of use for solving the MIP or if they can even be part of an integer feasible solution. Hence, any heuristic applied to find integer feasible solutions based on using such columns is always limited. In particular, if the root node integrality gap is large, there is a greater risk that the branching tree becomes large and many columns will need to be generated during the tree search—negatively impacting the effectiveness of the branch-and-price

algorithm.

In this paper, we present a matheuristic, in the form of a Large Neighbourhood Search (LNS) heuristic, which is designed to generate columns that are likely to be of use for near-optimally solving the MIP. The proposed LNS heuristic can preferably be used to complement other components in generic branch-price-and-cut solvers. It can be applied both in the root node to decrease the integrality gap already there, and also during the tree search to find better integer solutions to prune the tree. For our computational experiments, we have implemented the LNS heuristic in the generic branch-price-and-cut solver GCG (Bergner et al., 2015), which is an extension of the constraint integer programming solver SCIP (Gamrath et al., 2020). As the computational results show, our LNS heuristic performs especially well on challenging instances for which the integrality gap in the root node is large. Furthermore, it is also effective in finding high-quality integer feasible solutions for a broad range of problems, including instances with and without known underlying block diagonal structures.

1.1 Related work

Since branch-and-price has traditionally been viewed as an application-specific algorithm, the development of generic heuristics has been limited. The emergence of general purpose branch-price-and-cut solvers, such as BaPCod (Sadykov and Vanderbeck, 2021), Coluna.jl (Coluna, 2021), DIP (Galati et al., 2012), GCG (Bergner et al., 2015) and VRPSolver (Pessoa et al., 2020), has provided valuable platforms for the development and deployment of generic column generation-specific heuristics. However, this development is far behind that for general MIP solvers that are applied directly to compact formulations. In those, generic heuristics are capable of finding high-quality integer feasible solutions without any prior knowledge of the problem structure. A class of generic heuristics that are integral to the success of modern MIP solvers for compact formulations are LNS heuristics (Berthold, 2014).

Extending generic heuristics to the branch-and-price context is extremely challenging, see Sadykov et al. (2019) and Zhao et al. (2020). A major barrier is the difficulty defining neighbourhoods with respect to variables that are yet to be generated. Additionally, imposing upper bounds on variables, which is common practice in generic LNS heuristics, is well known to induce complicating changes to pricing problems in branch-and-price algorithms.

We have categorised our matheuristic as an LNS heuristic (described within a destroy-repair framework (Pisinger and Ropke, 2010)) and therefore the first part of our literature review is

focused on LNS heuristics and sub-MIPing for extended formulations. An important aspect of our heuristic design is to adapt the pricing problem with the aim to generate columns that are different to those typically found when relying on regular LP pricing. The second part of the literature review therefore addresses other work with such intentions.

1.1.1 LNS and sub-MIPing for extended formulations

LNS heuristics have been developed and deployed both as components within MIP solvers and as stand-alone solution methods. A central component of LNS heuristics, regardless of the method of deployment, is the formation of an auxiliary problem with a restricted feasible region, also referred to as a sub-MIP. The restriction of the feasible region is achieved through the addition of constraints that are typically in the form of variable fixings, but they may also be more general, such as local branching constraints (Fischetti and Lodi, 2003). In general, it is desired that the auxiliary problem has a feasible region that makes the problem *easy* to solve while containing high-quality primal solutions. As indicated above, the standard methods for forming sub-MIPs cannot be directly adopted in a branch-and-price context.

An elementary form of auxiliary problem in a branch-and-price context is to define a MIP over the columns generated to solve the LP relaxation of the master problem. Solving this sub-MIP by branch-and-bound tends to be simpler than performing branch-and-price, and will hopefully find an integer feasible solution. This is referred to as the restricted master heuristic (Lübbecke and Puchert, 2012) or price-and-branch (Desrosiers and Lübbecke, 2011). Since the included columns are generated only to solve the LP relaxation, the potential of this heuristic is rather limited. In particular, there is no guarantee that it will find a feasible solution (Lübbecke and Desrosiers, 2005).

Previous work related to more sophisticated heuristics to be applied in general-purpose branch-price-and-cut solvers has involved adaptations of generic heuristics available in MIP solvers. Successful examples of LNS heuristics for branch-and-price includes adaptations of the feasibility pump (Pesneau et al., 2012), crossover (Lübbecke and Puchert, 2012), and diving heuristics (Joncour et al., 2010; Sadykov et al., 2019). Most importantly, these generic heuristics demonstrate the increased potential of finding high-quality integer solutions when executing a column generation scheme during the solution of LNS auxiliary problems for extended formulations.

A technical challenge of LNS heuristics for branch-and-price is that the addition of con-

straints to form the auxiliary problem potentially induces changes that may not be amenable to the pricing problem solver. As highlighted in Pesneau et al. (2012) and Sadykov et al. (2019), the use of variable upper bounds or the addition of general constraints introduce additional dual variables that must be taken into account in the pricing problem for the correctness of the column generation algorithm. This imposes significant limitations on the restrictions that can be used to form an LNS auxiliary problem within branch-and-price algorithms. This is a particular concern when developing LNS heuristics using the destroy-and-repair framework, since the destroy step typically imposes upper bounds to fix variables to zero.

The issues raised by Pesneau et al. (2012) and Sadykov et al. (2019) related to the formation of LNS heuristic auxiliary problems are not encountered in the generic LNS heuristics presented in Lübbecke and Puchert (2012). The LNS heuristics discussed in Lübbecke and Puchert (2012) exploits the feature of GCG where the pricing problem is formulated and solved as a MIP. Thus, any modifications to the pricing problem that are required as a result of restrictions imposed to form the auxiliary problem are simply accounted for by changes to the corresponding pricing problem MIP. This feature of GCG is also exploited in the LNS heuristic developed in this paper—avoiding the limitations of LNS heuristics encountered in Pesneau et al. (2012) and Sadykov et al. (2019).

A major feature of our proposed heuristic, which further avoids the issues raised by Pesneau et al. (2012) and Sadykov et al. (2019), is that variable fixings imposed in the destroy step are handled in a way that does not affect the feasible set of the pricing problem. Instead, the restrictions imposed on the variables of the extended formulation are accounted for with respect to their impact in the set covering, packing or partitioning constraints. Based on this impact, the objective coefficients of the pricing problem are modified to favour columns that comply with the variable fixings. As such, the developed pricing scheme can be applied in general branch-and-price algorithms.

An important characteristic of the generic LNS heuristics developed in Pesneau et al. (2012), Lübbecke and Puchert (2012) and Sadykov et al. (2019) is that the generation of columns is performed to solve the LP relaxation of the auxiliary problem. Therefore, the generated columns are still only suitable for solving LP relaxations and the aspect of integrality is then delegated to the branching scheme. The modifications to the pricing problem proposed in this paper addresses this limitation by developing a pricing scheme that searches for columns that are expected to be part of integer feasible solutions.

A method that extends the heuristic search beyond the columns considered within a restricted master heuristic, but without the involvement of a branch-and-bound tree, is the SearchCol strategy (Alvelos et al., 2010, 2014). This method iterates between column generation for an LP relaxation of a restricted master problem to find profitable columns and a metaheuristic search on the available columns. The outcome of the latter is used to select restrictions to impose on the pricing problems in the next iteration with the purpose to guide and diversify the search for promising columns.

Extensions to the restricted master heuristic, that guarantee that an optimal solution of the MIP can be found, have been presented by Sweeney and Murphy (1979), Baldacci et al. (2006), and Baldacci et al. (2008). The central idea of the presented approaches (Sweeney and Murphy, 1979; Baldacci et al., 2006, 2008) is to generate a set of columns for the restricted master problem such that a subset of which is guaranteed to form an integer optimal solution. This solution can then be found by performing branch-and-bound on the restricted master problem using only the generated columns. A general description of optimality conditions for this context is found in Rönnberg and Larsson (2019). The benefit of this type of method is that no pricing is needed during the branch-and-bound process. The challenge is instead to generate a small-enough set of columns for the restricted master problem while still ensuring that this set contains an optimal solution.

Common for all the papers reviewed above is that column generation is used to solve an auxiliary problem, but pricing is still performed with respect to the LP relaxation. The primary differences between these papers are how this auxiliary problem is obtained and how the solution of the auxiliary problem is integrated with a branching scheme. Another related line of work involves to, in a more fundamental way, change the pricing scheme such that integrality is explicitly considered during the solving of the restricted master problem and while generating columns. Such approaches are reviewed in the following section.

1.1.2 Pricing for integrality

A method—not related to LNS heuristics and formulations of auxiliary problems—that is designed to increase the chances of finding columns that can be part of a high-quality integer solution within a branch-and-price context is presented in Bredström et al. (2014). They propose adding a surrogate constraint to the restricted master problem to tweak the column generation process used for solving the corresponding LP relaxation. A subgradient technique is then ap-

plied to a Lagrangian dual formulation of the master problem. The multipliers are heuristically adjusted to favour columns that are needed for forming an integer solution. This multiplier adjustment is constructed such that the method converges to a standard subgradient method, hence ensuring convergence to solving the LP relaxation of the restricted master problem.

Beyond the work of Bredström et al. (2014), we have only found that pricing for integrailty has been proposed outside the scope of branch-and-price. The first line of work uses the quasi-integrality property of problems with only set packing and set partitioning constraints (Trubin, 1969). This property of set packing and partitioning problems means that the edges of the convex hull of the integer feasible solutions are also edges of the polytope of feasible solutions to the LP relaxation of the problem. Thanks to this property it is possible to make simplex iterations that visit only integer extreme points. However, it is not straightforward to find the variables that yield these iterations. Methods for finding such pivots for compact formulations have been proposed in Trubin (1969), Thompson (2002), and Zaghrouti et al. (2014). By making special adaptations of the pricing problem it is possible to extend such methods to generate columns that can be used to pivot between integer solutions as proposed in Rönnberg and Larsson (2009, 2014) and Tahir et al. (2019).

Another line of work, with an alternative view on pricing for integer programs, is proposed in Zhao et al. (2020). Based on the optimality conditions in Larsson and Patriksson (2006), Zhao et al. (2020) formulate a pricing problem designed to greedily find *one* column that, when replacing a column in an integer feasible solution to a restricted master problem, is likely to yield an improved integer feasible solution. The focus of the paper was the theoretical derivation and justification that the pricing problem has these properties. The theoretical results of Zhao et al. (2020) were supported by some computational illustrations based on a simple heuristic implementation. The design and implementation of a computationally competitive method were left for future work. The theoretical results in Zhao et al. (2020) and Larsson and Patriksson (2006) are a part of the foundation for the method proposed in this paper. Because of their importance, and the need for notation to discuss them further, they are addressed more extensively among the preliminaries in Section 2.

1.2 Outline and contribution

This paper proposes a column generation-based matheuristic for MIP that can be applied to any problem where the master problem includes set covering, packing or partitioning constraints.

Our design is based on the destroy-and-repair LNS framework (Pisinger and Ropke, 2010) and it extends the work in Zhao et al. (2020) to perform pricing for integrailty based on the global optimality conditions presented in Larsson and Patriksson (2006). Section 2 reviews this previous work after introducing the relevant notation.

In the proposed LNS heuristic, the destroy method forms a partial solution by removing columns from a current primal feasible solution to a restricted master problem. The repair method consists of two steps: The first is to execute a customised repair pricing scheme and the second is to solve a repair problem to search for an improved integer solution. The novelty in this design is the repair pricing scheme that carefully selects columns to increase the chances of a successful outcome of solving the repair problem. The design of the LNS heuristic and the theoretical derivation of the repair pricing scheme is presented in Section 3.

The proposed heuristic has been implemented to complement the suite of heuristics within the branch-price-and-cut solver GCG (Bergner et al., 2015). This integration is addressed in Section 4. While the general destroy-and-repair principles underlying this heuristic can be applied as a stand-alone procedure, the integration with GCG brings many challenges and benefits. This paper will highlight the aspects that must be considered when balancing the exploration of integer feasible solutions against proving optimality within a general-purpose branch-price-and-cut solver.

The integration with GCG is used to show the general nature of the proposed heuristic and its wide applicability to many different problem types. This is supported by the results from the two types of computational evaluations presented in Section 5. First, a set of computational experiments to investigate key performance features is performed using publicly available instances originating from problem types with known block diagonal structures. A second set of experiments makes use of the automatic structure detection of GCG. Using a set of general instances from MIPLIB 2017 (Gleixner et al., 2021), GCG is applied to detect structures, perform a Dantzig-Wolfe reformulation and then solve the instances by branch-price-and-cut. The inclusion of the proposed heuristic within GCG is shown to significantly improve the effectiveness of branch-price-and-cut on both instance test sets. The paper is ended with some concluding comments in Section 6. With its inclusion in the distribution of GCG, the proposed LNS heuristic will be publicly available to the academic community.

2 Preliminaries

This section introduces notation and provides a foundation for the theoretical contributions of this paper. In particular, we review the results from Larsson and Patriksson (2006) and Zhao et al. (2020) that are used in Section 3.

2.1 Notation

Consider the binary program

$$
\begin{aligned}
\text{[MP]} \qquad \min \quad & \sum_{j \in \mathcal{J}} c_j \lambda_j, \\
\text{s.t.} \quad & \sum_{j \in \mathcal{J}} a_{ij} \lambda_j \geq 1, \quad i \in I^{\mathrm{c}}, \\
& \sum_{j \in \mathcal{J}} a_{ij} \lambda_j \leq 1, \quad i \in I^{\mathrm{p}}, \\
& (\lambda_j)_{j \in \mathcal{J}} \in \mathcal{L} \subseteq \{0,1\}^{|\mathcal{J}|},
\end{aligned}
$$

which is assumed to be an extended formulation derived by applying the discretisation approach (Johnson, 1989; Vanderbeck, 1994) to an original compact formulation of a binary program. As such, [MP] is referred to as a master problem. To simplify notation, we have omitted the details of the decomposition that results in this extended formulation. Instead, we refer to standard derivations such as the one found in Lübbecke and Desrosiers (2005) and note that for binary programs, discretisation and convexification coincide.

The columns (c_j, a_j), $a_j = (a_{ij})_{i \in I}$, $j \in \mathcal{J}$, are implicitly given as elements of a set $\mathcal{A} \subseteq \mathbb{R} \times \{0,1\}^{|I|}$ and they correspond to feasible solutions to a pricing problem. The explicit constraints are of set covering and set packing type and indexed by the sets I^{c} and I^{p}, respectively. In our formulation, a set partitioning constraint is represented by including both its covering and packing counterparts in [MP]. Further, let $I = I^{\mathrm{c}} \cup I^{\mathrm{p}}$ index the complete set of explicit constraints of [MP].

Before introducing the implicit constraints given by the set $\mathcal{L}$, some further notation is needed. It is assumed that the original problem can have a block diagonal structure. As such, the pricing problem separates into independent pricing problems. Let the set K index the independent pricing problems. To take into account that some of the pricing problems can be identical, the set K is partitioned into subsets—the set Q indexes these subsets. Then, for $q \in Q$, let the set K_q index pricing problems that are identical. Hence, there is a pricing problem

for each $q \in Q$ and this problem can generate columns for the identical pricing problems indexed by K_q. Let the complete set of columns for pricing problem $q \in Q$ be indexed by the set $\mathcal{J}_q$. Using this notation, the implicit constraints of [MP] are defined by

$$\mathcal{L} = \{\lambda_j \in \{0,1\}, j \in \mathcal{J} : \sum_{j \in \mathcal{J}_q} \lambda_j = |K_q|, q \in Q\}.$$

Further, let the set that implicitly defines the columns (c_j, a_j), $j \in \mathcal{J}_q$, be denoted by $\mathcal{A}_q$.

Next we will introduce the two types of reduced costs that are of relevance when generating columns in our setting. Let $u_i \geq 0$, $i \in I$, be dual multipliers associated with the explicit constraints of [MP]. By performing a Lagrangian relaxation of these constraints, the corresponding Lagrangian reduced costs are

$$\tilde{c}_j = c_j - \sum_{i \in I^{\mathrm{c}}} u_i a_{ij} + \sum_{i \in I^{\mathrm{p}}} u_i a_{ij}, \ j \in \mathcal{J}.$$

A pricing problem for $q \in Q$ with respect to this partial Lagrangian relaxation is

$$\begin{array}{lrll} [\mathrm{CG}_q] & \tilde{c}^q \ = & \min & c - \sum_{i \in I^{\mathrm{c}}} u_i a_i + \sum_{i \in I^{\mathrm{p}}} u_i a_i \\ & & \text{s.t.} & (c, a) \in \mathcal{A}_q. \end{array}$$

The corresponding LP reduced costs of the columns are derived as follows. Let [MP-LP] denote the LP relaxation of [MP], where $\lambda_j \in \{0,1\}$, $j \in \mathcal{J}$ is replaced by $\lambda_j \geq 0$, $j \in \mathcal{J}$. Let the multipliers $u_i \geq 0$, $i \in I$ take the role of the LP dual variables of the explicit constraints of [MP-LP]. We introduce v_q as the LP dual variables associated with the constraints $\sum_{j \in \mathcal{J}_q} \lambda_j = |K_q|$, $q \in Q$ in [MP-LP]. The LP reduced costs of the columns are

$$\bar{c}_j = c_j - \sum_{i \in I^{\mathrm{c}}} u_i a_{ij} + \sum_{i \in I^{\mathrm{p}}} u_i a_{ij} - |K_q| v_q, \ j \in \mathcal{J}_q, \ q \in Q.$$

A master problem restricted to include only a subset of columns, indexed by $J_q \subseteq \mathcal{J}_q$ for each subset $q \in Q$ of identical pricing problems, is referred to as a Restricted Master Problem [RMP]. Let $J = \cup_{q \in Q} J_q$ index the full set of columns at hand in [RMP], and note that it typically holds that $|J| \ll |\mathcal{J}|$. Let [RMP-LP] denote the LP relaxation of [RMP], where $\lambda_j \in \{0,1\}$, $j \in J$ is replaced by $\lambda_j \geq 0$, $j \in J$.

2.2 Greedy integer programming column generation

The method presented in this paper will use a pricing problem inspired by the Greedy Integer Programming Column Generation (GIP-CG) subproblem introduced in Zhao et al. (2020). To

establish a relation to this previous work and to present a foundation for the results to be presented in this paper, we begin with a derivation of a GIP-CG pricing problem for [MP]. Following the derivation, we give a brief description of the theoretical justifications for its design.

The work in Zhao et al. (2020) addresses a setting where a destroy-and-repair heuristic is used to find improving integer solutions to a restricted master problem. Each iteration of the heuristic involves removing one column and then immediately adding one new column to a current, not necessarily feasible, integer solution. Such current integer solution is simply defined by a set of columns that are active in the solution. The new column to add is generated and tailored to match the so-called partial solution, which is obtained after a column has been removed. Such partial solution is defined by the index sets $\hat{J}_q \subseteq J_q$, $q \in Q$, chosen such that $|\hat{J}_q| \leq |K_q|$ and $\sum_{q \in Q} |\hat{J}_q| < |K|$ hold. The partial solution is referred to by the index set $\hat{J} = \cup_{q \in Q} \hat{J}_q$.

The GIP-CG pricing problem in Zhao et al. (2020) was derived based on the global optimality conditions presented in Larsson and Patriksson (2006). Its objective comprises two parts: The first is related to the quality of the resulting master problem solution. The other concerns the feasibility in the restricted master problem. The formulation of the latter part, which is critical for finding high-quality integer solutions, depends on the problem structure. The general description in Zhao et al. (2020) does not prescribe how to do this for general integer programming problems. To derive a formulation for [MP] in our paper, the following notation is introduced, which indicates the feasibility of the explicit constraints of [RMP] in a partial solution given by the set $\hat{J}$. For the covering constraints, let the set of rows that are not covered in a partial solution be indexed by the set

$$\hat{I}^{\mathrm{c}} = \{i \in I^{\mathrm{c}} : \sum_{j \in \hat{J}} a_{ij} = 0\}.$$

To reach feasibility with respect to these rows will require the use of columns that cover these rows. For the packing constraints, let the set of rows where the partial solution already fills or even violates the packing constraints be indexed by the set

$$\hat{I}^{\mathrm{p}} = \{i \in I^{\mathrm{p}} : \sum_{j \in \hat{J}} a_{ij} \geq 1\}.$$

To maintain feasibility with respect to the rows that are fulfilled with equality, and to not create more infeasibility with respect to the rows that are already violated, only columns that do not

cover these rows can be used to repair the partial solution. The following proposition shows how to restate a GIP-CG in our setting.

Proposition 1. *Let the set $\hat{J}$ index columns that defines a partial solution to* [RMP] *and let $\bar{u}_i \geq 0$, $i \in I$, be a dual solution to* [RMP-LP]. *For a given $\gamma \in [0,1]$ and an $M \geq 0$, an optimal solution to* GIP-CG *as defined in (Zhao et al., 2020, p. 672) can be found by solving the problem* [GIP]

$$\min_{q \in Q : |\hat{J}_q| < |K_q|} \min_{j \in \mathcal{J}_q} \left\{ c_j - \sum_{i \in I^c} \gamma u_i a_{ij} + \sum_{i \in I^p} \gamma u_i a_{ij} - \sum_{i \in \hat{I}^c} M a_{ij} + \sum_{i \in \hat{I}^p} M a_{ij} \right\}.$$

Proof. We begin by translating the result in Zhao et al. (2020) to the notation and setting used in our paper. In the generic definition of GIP-CG (Zhao et al., 2020, p. 672), a partial solution is given. The columns of the corresponding partial solution in the current paper is indexed by the set $\hat{J}$. The general derivation in Zhao et al. (2020) considers the case of generating one new column to be added to a restricted master problem such that a violation of the implicit constraints is not introduced. The corresponding assumption in our setting is that for the partial solution, there exists a $\hat{q} \in Q$ such that $|\hat{J}_{\hat{q}}| \leq |K_{\hat{q}}| - 1$ holds. Further, GIP-CG is defined only for the columns that will not increase the violation of the implicit constraints of [RMP]. This general requirement translates to our setting as only considering columns $j \in \mathcal{J}_q$ for $q \in Q$ such that $|\hat{J}_q| < |K_q|$ hold. Using the notation of the current paper, and also including $\leq$-constraints by changing the appropriate signs, GIP-CG can be stated as

$$\min_{q \in Q : |\hat{J}_q| < |K_q|} \min_{j \in \mathcal{J}_q} \left\{ c_j - \sum_{i \in I^{\mathrm{c}}} \gamma u_i a_{ij} + \sum_{i \in I^{\mathrm{p}}} \gamma u_i a_{ij} + \right.$$
$$\left. M \sum_{i \in I^{\mathrm{c}}} \max \left\{ 0, 1 - \sum_{j' \in \hat{J}} a_{ij'} - a_{ij} \right\} + M \sum_{i \in I^{\mathrm{p}}} \max \left\{ 0, a_{ij} + \sum_{j' \in \hat{J}} a_{ij'} - 1 \right\} \right\}.$$

Note that the first three terms are the same as those in [GIP] and that the last two terms penalise infesibility with respect to the explicit constraints in the restricted master problem. Consider these last two terms further. Since $\sum_{j \in \hat{J}} a_{ij} = 0$ for $i \in \hat{I}^{\mathrm{c}}$ and $\sum_{j \in \hat{J}} a_{ij} \geq 1$ for $i \in I^{\mathrm{c}} \setminus \hat{I}^{\mathrm{c}}$, it holds that

$$\sum_{i \in I^{\mathrm{c}}} \max \left\{ 0, 1 - \sum_{j' \in \hat{J}} a_{ij'} - a_{ij} \right\} = \sum_{i \in \hat{I}^{\mathrm{c}}} (1 - a_{ij}).$$

Since $\sum_{j\in\hat{J}} a_{ij} \geq 1$ for $i \in \hat{I}^{\mathrm{p}}$ and $\sum_{j\in\hat{J}} a_{ij} = 0$ for $i \in I^{\mathrm{p}} \setminus \hat{I}^{\mathrm{p}}$, it also holds that

$$\sum_{i\in I^{\mathrm{p}}} \max\left\{0, a_{ij} + \sum_{j'\in\hat{J}} a_{ij'} - 1\right\} = \sum_{i\in\hat{I}^{\mathrm{p}}} (a_{ij} + \sum_{j'\in\hat{J}} a_{ij'} - 1).$$

Hence, the two terms that penalise infesibility in GIP-CG can equivalently be expressed as

$$-\sum_{i\in\hat{I}^{\mathrm{c}}} M a_{ij} + \sum_{i\in\hat{I}^{\mathrm{p}}} M a_{ij} + \sum_{i\in\hat{I}^{\mathrm{c}}} M + \sum_{i\in\hat{I}^{\mathrm{p}}} M(\sum_{j'\in\hat{J}} a_{ij'} - 1),$$

which differs only by constant terms from the last two terms in [GIP]. Since GIP-CG and [GIP] are defined over the same sets of columns and their objective functions differ only by constant terms, an optimal solution to GIP-CG can be found by solving [GIP]. □

The objective of GIP-CG has one component, given as the last two terms in [GIP], that, in a l_1-norm sense, penalises columns that will violate the explicit constraints in [RMP]. The purpose of this component is to promote columns that can be part of a primal feasible solution to [RMP].

The other component in the objective of GIP-CG promotes columns that are likely to yield an improvement of the objective value if included in a solution to [RMP]. This component is given by the first three terms and it differs from the objective function of [CG$_q$], $q \in Q$, only by the parameter γ. We will not provide a full derivation that explains the inclusion of the parameter γ; instead, we refer to Sections 2.1–2.2 in Zhao et al. (2020) and provide here only a brief discussion and explain how the parameter can be chosen depending on the properties of the problem to be solved.

In Zhao et al. (2020), GIP-CG is derived from the optimality conditions presented in Larsson and Patriksson (2006). These optimality conditions extend the general primal-dual optimality conditions (see Theorem 2 in Zhao et al. (2020) and Theorem 6.2.5 in Bazaraa et al. (1993)) to be valid also for problems with a positive dual gap. A key facilitator for the extension in (Larsson and Patriksson, 2006) is to introduce the quantity δ-complementarity that accounts for the deviation from complementarity in the explicit constraints; it can be expressed as

$$\delta\left((\lambda_j)_{j\in\mathcal{J}}, (u_i)_{i\in I}\right) = \sum_{i\in I^{\mathrm{c}}} u_i \left(\sum_{j\in\mathcal{J}} a_{ij}\lambda_j - 1\right) + \sum_{i\in I^{\mathrm{p}}} u_i \left(1 - \sum_{j\in\mathcal{J}} a_{ij}\lambda_j\right)$$

in our setting. Complementarity holds in an optimal solution $(\lambda_j^*)_{j\in\mathcal{J}}, (u_i^*)_{i\in I}$ if $\delta\left((\lambda_j^*)_{j\in\mathcal{J}}, (u_i^*)_{i\in I})\right) = 0$, e.g. as for extreme point solutions to a linear program. When complementarity is expected

to hold in an optimal solution, the derivation in Zhao et al. (2020) shows that $\gamma = 1$ should be used in GIP-CG. This corresponds to using the standard Lagrangian reduced costs as in [CG$_q$], $q \in Q$. However, if complementarity is not expected to hold, as can be the case for integer programs, it is reasonable to use a $\gamma < 1$. The exact value to choose for γ is not known, as it depends on the degree of δ-complementarity for the specific instance of a problem.

The pricing problem that will be described for the heuristic introduced in Section 3 will, similar to above, have an objective function that comprises two parts. The first part will have to do with the quality of the solution and this will also consider δ-complementarity, but in another sense than for GIP-CG. The second part of the objective will concern feasibility, but—again—in a different way than described above.

3 IPColGen – An integer programming column generation matheuristic

The core component of the LNS heuristic presented in this section is a repair method that uses a novel pricing scheme specifically designed to generate columns that repair a partial integer solution to [RMP]. This section introduces the heuristic design and provides a theoretical derivation and justification for the design of the pricing scheme. To aid the exposition, Appendix A presents a numerical example that can be read alongside this section. The proposed LNS heuristic is intended to be applied in a generic branch-price-and-cut solver but it is presented, and can also be used, as a stand-alone procedure. The integration with the generic branch-price-and-cut solver GCG is addressed separately in Section 4.

3.1 Method framework

Our method will be described within the LNS framework, see Pisinger and Ropke (2010) for details. LNS is a metaheuristic search method introduced in Shaw (1998) where, in each iteration, a large neighbourhood is explored by first destroying a current solution and then repairing it. The general LNS framework is displayed in Algorithm 1. The key design considerations are the definitions of the destroy and repair methods as these implicitly define the neighbourhoods and direct the search for feasible solutions.

Our LNS heuristic is applied to an [RMP] where the repair method uses a customised pricing scheme and solves a repair problem to find improving integer solutions to [MP]. This customised

Algorithm 1 Large-neighbourhood search

Require: A primal feasible solution

Ensure: An updated primal feasible solution

```
Let the current solution be the input primal feasible solution
Let the best solution be the current solution
repeat
    Apply a destroy method to the current solution
    Apply a repair method to the destroyed solution
    if Acceptance criterion is met then
        Let the current solution be the repaired solution
    end if
    if The current solution is better than the best solution then
        Let the best solution be the current solution
    end if
until Stop criterion is met
```

pricing scheme is a major difference from previous work (Pesneau et al., 2012; Lübbecke and Puchert, 2012; Sadykov et al., 2019) on generic LNS heuristics for branch-and-price. To define the destroy and repair methods for a given [RMP], the following notation is introduced.

- A **primal feasible solution** to [RMP] is referred to by the columns that are active in the solution. For each $q \in Q$, let the set $J_q^{\text{IP}} \subseteq J_q$ index all active columns from subset q of identical pricing problems, that is, the columns $j \in J_q$ for which it hold that $\lambda_j = 1$. Further, use the set $J^{\text{IP}} = \cup_{q \in Q} J_q^{\text{IP}}$ to index all active columns and the corresponding primal feasible solution.

- A **partial solution** to [RMP] is defined by a set of columns that is obtained by removing columns from a primal feasible solution. Let the sets $\hat{J}_q \subseteq J_q^{\text{IP}}$, $q \in Q$, index columns chosen such that $|\hat{J}_q| \leq |K_q|$ and $\sum_{q \in Q} |\hat{J}_q| < |K|$ hold. A partial solution is then referred to by the index set $\hat{J} = \cup_{q \in Q} \hat{J}_q$. Note that this definition differs slightly from that presented in Section 2.2, where the partial solution did not need to originate from a feasible solution.

- Denote an LP **dual solution** to [RMP-LP] by $\bar{u}_i$, $i \in I$.

The **destroy method** forms a partial solution $\hat{J}$ by removing columns from a current primal feasible solution J^{IP}. The **repair method** consists of two steps: The first is to generate a set

of repair columns indexed by the set $J^{\mathrm{R}} = \cup_{q \in Q} J^{\mathrm{R}}_q$, where $J^{\mathrm{R}}_q \subseteq \mathcal{J}_q$, $q \in Q$, and the second is to solve the repair problem

$$
\begin{aligned}
\text{[REP]} \qquad \min \quad & \sum_{j \in J^{\mathrm{R}}} c_j \lambda_j + \sum_{j \in \hat{J}} c_j, \\
\text{s.t.} \quad & \sum_{j \in J^{\mathrm{R}}} a_{ij} \lambda_j \geq 1 - \sum_{j \in \hat{J}} a_{ij}, \quad i \in I^{\mathrm{c}}, \\
& \sum_{j \in J^{\mathrm{R}}} a_{ij} \lambda_j \leq 1 - \sum_{j \in \hat{J}} a_{ij}, \quad i \in I^{\mathrm{p}}, \\
& \sum_{j \in J^{\mathrm{R}}_q} \lambda_j = |K_q| - |\hat{J}_q|, \quad q \in Q, \\
& \lambda_j \in \{0, 1\}, j \in J^{\mathrm{R}} \cup J.
\end{aligned}
$$

If [REP] has an optimal solution with a better objective value than that of the current primal feasible solution defined by J^{IP}, the solution to [REP]—together with the columns indexed by $\hat{J}$—becomes the new primal feasible solution. This is the acceptance criterion on line 6 in Algorithm 1. If the new solution is not accepted, the procedure is repeated by further destroying the partial solution and then the repair method is applied again. The stop criterion on line 12 in Algorithm 1 is fulfilled after a prespecified number of iterations or if a local optimum is found.

A local optimum, with respect to the neighbourhood defined by our destroy and repair methods, is reached when the partial solution becomes empty after applying the destroy method. This happens after repeatedly applying the destroy and repair methods without obtaining an improved feasible solution. Note that it holds that an optimal solution to the original problem will be a local optimum in this sense. This follows from the fact that when an optimal solution is destroyed, the repair method can never find a better solution. Thus, the destroy method will be repeatedly applied until the partial solution becomes empty.

Given a partial solution $\hat{J}$, the quality of the best primal feasible solution to [RMP] that can be obtained by solving [REP] depends on the repair columns, indexed by J^{R}. The next two sections provide a derivation of how we generate these columns. First, Section 3.2 gives a theoretical justification for the design of our repair pricing scheme. Section 3.3 then discusses how to choose values for the parameters included in the repair pricing objective.

3.2 Repair pricing scheme for [REP]

The repair pricing scheme is designed to generate a set of repair columns, indexed by J^{R}, to be included in [REP] with the goal that the solution to [REP] is an improved integer feasible

solution for [RMP]. Note that an important difference to the setting in Zhao et al. (2020) is that instead of greedily choosing a single best possible column to replace a removed one we want to generate a set of columns and then solve [REP] to find a new solution to [RMP].

Ideally, we would like to generate a set of columns that are active in an optimal solution to [REP] when it is solved over the set of columns indexed by $\mathcal{J} \setminus \hat{J}$. Such a set of columns, termed ideal columns, would yield a best possible new solution to [RMP] and is defined as follows.

Definition 1. *Let [REP] be defined for a partial solution indexed by $\hat{J}$ and for the set of columns indexed by $J^R = \mathcal{J} \setminus \hat{J}$. A set of columns indexed by J^{R^*} is an* ideal set of columns *for [REP] if it holds that the solution with $\lambda_j = 1$, $j \in J^{R^*}$ and $\lambda_j = 0$, $j \in J^R \setminus J^{R^*}$ is optimal in [REP].*

This set of columns cannot be found without evaluating all possible columns to include in [REP]. Therefore, our strategy is to instead generate columns that are chosen such that they are likely to be part of a high-quality solution to [REP]. This generation is made iteratively, adding one new column at a time. We refer to this procedure as a repair pricing scheme, and it relies on the following property of the set J^{R^*}.

Remark 1. *The set J^{R^*} can be used to index the ideal columns even if these columns are not known. This follows since, in any solution to [REP] for a fixed partial solution, the last set of constraints in [REP] implies that the number of columns from pricing problem $q \in Q$ is $|K_q| - |\hat{J}_q|$.*

Our repair pricing scheme generates a set of columns indexed by L_j for each $j \in J^{\mathrm{R}^*}$ so that the total set of columns generated after applying the scheme, for a given partial solution, is indexed by $L = \cup_{j \in J^{\mathrm{R}^*}} L_j$. Then, [REP] is defined for $J^{\mathrm{R}} = L$. In iteration $l = 0, \ldots, |L| - 1$ of the repair pricing scheme, let $\hat{L}_{jl} \subseteq L_j$ index the set of generated columns. Note that $L_j = \hat{L}_{jl}$ for $l = |L| - 1$, $j \in J^{\mathrm{R}^*}$.

The derivation of our repair pricing scheme is based on considering the contributions of the columns in $\hat{L}_{jl}$, $l = 0, \ldots, |L| - 1$, $j \in J^{\mathrm{R}^*}$, in an average sense. Therefore, we begin by making the following definition.

Definition 2. *Let J index a set of columns. The* average column *for the set J is the column $\frac{1}{|J|} \sum_{j \in J} (c_j, a_j)$.*

The aim of the repair pricing scheme is that the average column for the set L_j approximates the ideal column indexed by $j \in J^{\mathrm{R}^*}$ so that $(c_j, a_j) \approx \frac{1}{|L_j|} \sum_{j' \in L_j} \left(c_{j'}, a_{j'}\right)$ holds.

Using this concept of an average column means that we do not consider which of the generated columns will actually be active in a solution to [REP]. What we do consider is that the generated columns have profitable properties on average. This is advantageous since the set of active columns can only be known after [REP] has been solved for the full set L. To iteratively improve this approximation during a round of repair pricing, we consider the contribution from the columns generated up to the current round and let these influence the further search as follows.

The approximation of the ideal columns can be considered as trying to mimic important properties of these columns. Feasibility in the explicit constraints of [MP] is a property that is fundamental in the search for integer feasible solutions. Let $c^* = \sum_{j \in J^{\mathrm{R}*}} c_j$ and use the notation

$$a_i^* = \sum_{j \in J^{\mathrm{R}*}} a_{ij}, \ i \in I$$

to refer to the contribution that a set of ideal columns would make to the left-hand side of the constraints for $i \in I^{\mathrm{c}}$ and $i \in I^{\mathrm{p}}$ in [REP]. To achieve feasibility with respect to these constraints, terms are included in the pricing problem objective to penalise infeasibility. In the following, we provide a derivation of these terms and highlight the differences to the approach described in Section 2.2.

For the covering constraints, let the set of rows that are not covered in a partial solution be indexed by the set

$$\hat{I}^{\mathrm{c}0} = \{i \in I^{\mathrm{c}} : \sum_{j \in \hat{J}} a_{ij} = 0\}$$

and let the set of rows that are covered in a partial solution be indexed by the set

$$\hat{I}^{\mathrm{c}1} = \{i \in I^{\mathrm{c}} : \sum_{j \in \hat{J}} a_{ij} \geq 1\}.$$

Similarly, for the packing constraints, let the set of rows that are not filled by the partial solution be indexed by the set

$$\hat{I}^{\mathrm{p}0} = \{i \in I^{\mathrm{p}} : \sum_{j \in \hat{J}} a_{ij} = 0\}$$

and let the set of rows where the partial solution already satisfies the packing constraints with equality be indexed by the set

$$\hat{I}^{\mathrm{p}1} = \{i \in I^{\mathrm{p}} : \sum_{j \in \hat{J}} a_{ij} = 1\}.$$

Using this notation, a necessary property of the ideal columns can be described as follows.

Lemma 1. *Let* [REP] *be defined for a partial solution* $\hat{J}$ *such that* [REP] *is feasible. If the set* J^{R^*} *index columns that define an optimal solution to* [REP]*, it holds that*

$$\begin{aligned} a_i^* &\geq 1,\ i \in \hat{I}^{\mathrm{c0}}, \\ a_i^* &\geq 0,\ i \in \hat{I}^{\mathrm{c1}}, \\ a_i^* &\leq 1,\ i \in \hat{I}^{\mathrm{p0}}, \\ a_i^* &= 0,\ i \in \hat{I}^{\mathrm{p1}}. \end{aligned}$$

Proof. Together, the columns indexed by the sets $\hat{J}$ and J^{R^*} constitute a feasible solution to [REP] and therefore $\sum_{j\in\hat{J}} a_{ij} + \sum_{j\in J^{\mathrm{R}^*}} a_{ij} \geq 1$, $i \in I^{\mathrm{c0}} \cup I^{\mathrm{c1}}$ and $\sum_{j\in\hat{J}} a_{ij} + \sum_{j\in J^{\mathrm{R}^*}} a_{ij} \leq 1$, $i \in I^{\mathrm{p0}} \cup I^{\mathrm{p1}}$ hold. Then, the definition of the set $\hat{I}^{\mathrm{c0}}$ yields that $\sum_{j\in J^{\mathrm{R}^*}} a_{ij} \geq 1 - \sum_{j\in\hat{J}} a_{ij} = 1$, $i \in \hat{I}^{\mathrm{c0}}$, holds. Similarly, the definition of the set $\hat{I}^{\mathrm{p0}}$ yields that $\sum_{j\in J^{\mathrm{R}^*}} a_{ij} \leq 1 - \sum_{j\in\hat{J}} a_{ij} = 1$, $i \in \hat{I}^{\mathrm{p0}}$, holds. Further, for $i \in \hat{I}^{\mathrm{p1}}$ it holds that $\sum_{j\in\hat{J}} a_{ij} = 1$ implying that $\sum_{j\in J^{\mathrm{R}^*}} a_{ij} \leq 1 - \sum_{j\in\hat{J}} a_{ij} = 0$ also holds, which together with $a_{ij} \geq 0$, $j \in \mathcal{J}$, yields that $\sum_{j\in J^{\mathrm{R}^*}} a_{ij} = 0$. Lastly, since $a_{ij} \geq 0$, $j \in \mathcal{J}$, it holds that $\sum_{j\in J^{\mathrm{R}^*}} a_{ij} \geq 0$, $i \in \hat{I}^{\mathrm{c1}}$. □

The following lemma introduces the properties of a column that will later be used as the benchmark when assessing, in an average sense, the properties of columns that have been generated. This column is constructed such that if this is the column generated in each iteration of the repair pricing scheme, then, together the average columns for the sets of columns generated for each $j \in J^{\mathrm{R}^*}$ would yield the same objective value in [REP] as the ideal columns.

Lemma 2. *Let* [REP] *be defined for a partial solution indexed by* $\hat{J}$ *such that* [REP] *is feasible. For each* $j \in J^{R^*}$*, let* $\tilde{L}_j$ *index a set of identical columns with coefficients*

$$\frac{1}{|J^{R^*}|}\left(c^*, (a_i^*)_{i\in I}\right).$$

Let $\left(\tilde{c}_j, (\tilde{a}_j)_{i\in I}\right)$ *denote the average column for the set* $\tilde{L}_j$, $j \in J^{R^*}$. *The set of columns* $\left\{\left(\tilde{c}_j, (\tilde{a}_j)_{i\in I}\right)\right\}_{j\in J^{R^*}}$ *is an ideal set of columns for* [REP].

Proof. The result follows since

$$\sum_{j\in J^{\mathrm{R}^*}} \left(\tilde{c}_j, (\tilde{a}_j)_{i\in I}\right) = \sum_{j\in J^{\mathrm{R}^*}} \frac{1}{|\tilde{L}_j|} \sum_{j'\in\tilde{L}_j} \frac{1}{|J^{\mathrm{R}^*}|} \left(c^*, (a_i^*)_{i\in I}\right) = \left(c^*, (a_i^*)_{i\in I}\right).$$

□

Note that these identical columns typically cannot be attained in practice as their constraint coefficients are not necessarily binary.

Lemmas 1 and 2 state the properties of an ideal set of columns for [REP]. Note that in Lemma 1, the properties of a_{ij} for $i \in \hat{I}^{\mathrm{p1}} \cup \hat{I}^{\mathrm{c1}}$ translates to each individual ideal column, while for $i \in \hat{I}^{\mathrm{p0}} \cup \hat{I}^{\mathrm{c0}}$ the property concerns all the ideal columns jointly. To better illustrate how to mimic these two properties when generating columns for the sets L_j for each $j \in J^{\mathrm{R}^*}$, we separate them into static and dynamic feasibility properties, respectively. In the following, we begin by focusing on the static feasibility properties and thereafter the dynamic feasibility properties are addressed.

Theorem 1 (Static feasibility properties)**.** *If a column (c_j, a_j), $j \in L$, is included in a solution to* [REP] *it holds that*

$$a_{ij} \in \{0,1\},\ i \in \hat{I}^{\mathrm{c1}},$$
$$a_{ij} = 0,\ i \in \hat{I}^{\mathrm{p1}}.$$

Proof. The columns included in a solution to [REP] must constitute a feasible solution in the same sense as the ideal columns. The result follows from Lemma 1 together with the requirement that $a_{ij} \in \{0,1\}$, $i \in I$, $j \in \mathcal{J}$. □

The result from Theorem 1 means that columns where $a_i = 1$, $i \in \hat{I}^{\mathrm{p1}}$, must not be generated, since these can never be part of a feasible solution to [REP]. Therefore, in the formulation of the repair pricing problem below, there will be a large static penalty M to force $a_i = 0$, $i \in \hat{I}^{\mathrm{p1}}$. Furthermore, Theorem 1 states that no restriction is needed with respect to the constraints $i \in \hat{I}^{\mathrm{c1}}$. Note that the result of Theorem 1 also holds if a column (c_j, a_j), $j \in L$, is replaced by an average column for some set of columns.

For the remaining rows, indexed by $i \in \hat{I}^{\mathrm{p0}} \cup \hat{I}^{\mathrm{c0}}$, it is not sufficient to consider the feasibility requirements with respect to individual columns. Instead, we study the average column for the set of columns generated this far to wisely direct the further search.

Theorem 2 (Dynamic feasibility properties)**.** *For $j \in J^{R^*}$ and iteration $l = 0, \ldots, |L| - 1$ of the repair pricing scheme*

- *let $\hat{L}_{jl} \subseteq L_j$ index the sets of generated columns and denote the average column for the set $\hat{L}_{jl}$ by $(\hat{c}_{jl}, \hat{a}_{jl})$, and*
- *let $\tilde{L}_{jl} : |\tilde{L}_{jl}| = |\hat{L}_{jl}|$ index sets of identical ideal columns*

$$\frac{1}{|J^{R^*}|} \sum_{j' \in J^{R^*}} \left(c^*, (a_i^*)_{i \in I}\right)$$

for [REP] *and denote the average column for the set* $\tilde{L}_{jl}$ *by* $(\tilde{c}_{jl}, \tilde{a}_{jl})$.

If the contribution to the left-hand side of the constraints $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$ *in* [REP] *is the same for the columns* $(\hat{c}_{jl}, \hat{a}_{jl})$ *and* $(\tilde{c}_{jl}, \tilde{a}_{jl})$*, then it holds that*

$$\sum_{j \in J^{R^*}} \sum_{j' \in \hat{L}_{jl}} a_{ij'} \begin{cases} \geq \frac{1}{|J^{R^*}|} \sum_{j \in J^{R^*}} |\hat{L}_{jl}|, \ i \in \hat{I}^{\mathrm{c}0}, \\ \leq \frac{1}{|J^{R^*}|} \sum_{j \in J^{R^*}} |\hat{L}_{jl}|, \ i \in \hat{I}^{\mathrm{p}0}. \end{cases}$$

Proof. Since the contribution to the left-hand side of the constraints $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$ in [REP] is the same for the columns $(\hat{c}_{jl}, \hat{a}_{jl})$ and $(\tilde{c}_{jl}, \tilde{a}_{jl})$, $j \in J^{\mathrm{R}^*}$, $l = 0, \ldots, |L| - 1$, it holds that

$$\frac{1}{|\hat{L}_{jl}|} \sum_{j' \in \hat{L}_{jl}} a_{ij'} = \frac{1}{|\tilde{L}_{jl}|} \sum_{j' \in \tilde{L}_{jl}} \frac{a_i^*}{|J^{\mathrm{R}^*}|} \Leftrightarrow \sum_{j' \in \hat{L}_{jl}} a_{ij'} = \frac{|\hat{L}_{jl}| a_i^*}{|J^{\mathrm{R}^*}|}.$$

Taking the sum over $j \in J^{\mathrm{R}^*}$ and using Lemma 1 yields

$$\sum_{j \in J^{\mathrm{R}^*}} \sum_{j' \in \hat{L}_{jl}} a_{ij} = \sum_{j \in J^{\mathrm{R}^*}} \frac{|\hat{L}_{jl}| a_i^*}{|J^{\mathrm{R}^*}|} \begin{cases} \geq \frac{1}{|J^{\mathrm{R}^*}|} \sum_{j \in J^{\mathrm{R}^*}} |\hat{L}_{jl}|, \ i \in \hat{I}^{c0}, \\ \leq \frac{1}{|J^{\mathrm{R}^*}|} \sum_{j \in J^{\mathrm{R}^*}} |\hat{L}_{jl}|, \ i \in \hat{I}^{p0}, \end{cases}$$

and the conclusion follows. □

The inequality presented in Theorem 2 can hence be used to assess an average feasibility status of the generated columns in comparison to an ideal set of columns. By doing so, the penalty for violating these constraints can be dynamically adjusted depending on the columns that have been generated this far. For this purpose, let β_{il} denote the penalty for the row indexed by $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$ in iteration $l = 0, \ldots, |L| - 1$.

In line with the presented arguments, the **repair pricing problem** for the identical pricing problems indexed by $q \in Q$ in iteration $l = 0, \ldots, |L| - 1$ of the repair pricing scheme is formulated as

$$\begin{aligned} [\text{REP-CG}_{ql}] \quad \min \quad & c - \sum_{i \in I^{\mathrm{c}}} \gamma \bar{u}_i a_i + \sum_{i \in I^{\mathrm{p}}} \gamma \bar{u}_i a_i + \\ & + \sum_{i \in \hat{I}^{\mathrm{p}1}} M a_i - \sum_{i \in \hat{I}^{\mathrm{c}0}} \beta_{il} a_i + \sum_{i \in \hat{I}^{\mathrm{p}0}} \beta_{il} a_i \\ \text{s.t.} \quad & (c, a) \in \mathcal{A}_q. \end{aligned}$$

Note that the first three terms in the objective are the same as those for [GIP] in Proposition 1, even if the δ-complementarity that γ stems from has another origin than that for [GIP]. This will be further discussed in Section 3.3 and henceforth we will refer to γ as the **complementarity parameter**.

In [GIP], a penalty is used to force the constraint coefficients to have certain values. A major difference between [GIP] and [REP-CG$_{ql}$] is that the latter includes both a static penalty for properties that must be fulfilled by all generated columns and a dynamic penalty to guide the search to find columns that are likely to be part of a feasible solution to [REP].

3.3 Parameters in the repair pricing problem

The repair pricing problem includes parameters that need to be assigned appropriate values. The first is the complementarity parameter γ that was introduced for [GIP] in Proposition 1. In the context of [GIP], the value of γ corresponds to the amount of δ-complementarity to expect in [RMP] when greedily adding one new column. In this paper, and more specifically in the context of [REP-CG$_{ql}$], the δ-complementarity refers neither to [RMP] nor to [REP]. Instead, it refers to the effect that iteration $\hat{l}$ in the repair pricing scheme has on [REP].

In iteration $\hat{l}$, the average columns for the sets $\hat{L}_{j(\hat{l}-1)}$, $j \in J^{\mathrm{R}^*}$ are already known and some columns remain to be generated in iterations $l = \hat{l}+1, \ldots, |L|-1$. Note that during our repair pricing scheme, we always consider the average contributions from the columns that have been generated for each $j \in J^{\mathrm{R}^*}$. Therefore, the δ-complementarity after iteration $\hat{l}$ will be

$$\delta\left((\bar{u}_i)_{i\in I}\right) = \sum_{i\in I^{\mathrm{c}}} \bar{u}_i \left(\sum_{j\in J^{\mathrm{R}^*}} \frac{1}{|\hat{L}_{j\hat{l}}|} \sum_{j'\in \hat{L}_{j\hat{l}}} a_{ij'} - \left(1 - \sum_{j\in\hat{J}} a_{ij}\right)\right) + \sum_{i\in I^{\mathrm{p}}} \bar{u}_i \left(1 - \sum_{j\in\hat{J}} a_{ij} - \sum_{j\in J^{\mathrm{R}^*}} \frac{1}{|\hat{L}_{j\hat{l}}|} \sum_{j'\in\hat{L}_{j\hat{l}}} a_{ij'}\right).$$

Neither the average column for the sets $\hat{L}_{j(\hat{l}-1)}$, nor the average columns for the sets $\hat{L}_{j\hat{l}}$, $j \in J^{\mathrm{R}^*}$, are expected to have only binary coefficients. On the contrary, it can be expected that they are fractional. Hence, it is expected that the value of $\delta((\bar{u}_i)_{i\in I})$ is non-zero. Thus, the actual value of γ needs to be tuned with respect to the amount of δ-complementarity that is to be expected. The setting of γ in the practical application of the proposed heuristic is investigated computationally in Section 5.

The values of β_{il}, $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$, $l = 0, \ldots, |L|-1$, need to be chosen with the aim to generate columns that comply with the inequality in Theorem 2. In our scheme, we choose an initial value to assign to β_{i0}, $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$, and then we make updates and assign values to β_{il}, $i \in I^{\mathrm{c}0} \cup I^{\mathrm{p}0}$, $l = 1, \ldots, |L|-1$ depending on if the conditions given by Theorem 2 are satisfied. When determining how to make these updates, there is a trade-off between feasibility in [REP]

and the cost of the column. We propose to handle this trade-off by initialising with a small value of β_{i0}, $i \in I^{c0} \cup I^{p0}$, and then rather rapidly increase the penalty for an $i \in I^{c0} \cup I^{p0}$ when the inequality in Theorem 2 is violated and to, similarly, decrease it rapidly when the inequality is satisfied.

An important note about the objective of [REP-CG$_{ql}$] is that there can be two objective coefficients for the same a_i, $i \in I$, but with different origins. The coefficient $\gamma\bar{u}_i$ depend on the current solution to [RMP-LP] and if the constraint indexed by $i \in I$ is not active in the solution to [RMP-LP] then $\bar{u}_i = 0$. The other coefficient, either β_{il} or M, does instead depend on $\hat{J}$, the partial solution to [RMP]. Hence, the objective is influenced by an interplay between the dual solution to [RMP-LP] and the partial solution to [RMP] that is used.

In IPColGen, a set partitioning constraint is included through its covering and packing counterparts. During the repair pricing scheme, it is not likely that both of these constraints are complied with by the current average column. Instead, infeasibility is expected in one of them while there is a non-zero δ-complementarity in the other. It is likely that these roles will alternate between the two constraints. Thus, it is reasonable for δ to be non-zero for the set partitioning constraints during the execution of the repair pricing scheme.

4 Implementation and integration with GCG

The proposed LNS heuristic has been implemented within the generic branch-price-and-cut solver GCG (Bergner et al., 2015), which is an extension of the constraint integer programming solver SCIP (Gamrath et al., 2020). GCG is a generic MIP solver. As such, given a general MIP instance, GCG will automatically detect decomposable structures, perform a Dantzig-Wolfe reformulation using a selected structure and then solve the resulting reformulated problem using branch-price-and-cut. It is also possible to skip the detection stage by supplying a known decomposition structure. Importantly, GCG contains a suite of column generation-specific primal heuristics, such as those described by Lübbecke and Puchert (2012). A major contribution of this paper is the incorporation of IPColGen into this suite of primal heuristics to improve the ability of GCG to find high quality integer solutions for difficult MIP problems.

Since GCG is a generic solver, the column generation pricing problem is usually formulated and solved as a MIP instance—typically using SCIP. Solving the pricing problem as a MIP avoids many of the issues related to LNS heuristics for branch-and-price that are highlighted

by Pesneau et al. (2012) and Sadykov et al. (2019). However, it is important to note that since IPColGen only modifies the subproblem objective, and not the subproblem constraints, the issues raised by Pesneau et al. (2012) and Sadykov et al. (2019) are not encountered in any application of the proposed heuristic.

The IPColGen algorithm has a high computational cost resulting from the execution of multiple repairing pricing rounds and solving of the repair problem [REP]. Thus, the point at which IPColGen is executed during the node processing must be selected carefully. An adaptive procedure, based on the identification of the column generation tailing-off effect (Lübbecke and Desrosiers, 2005) and current optimality gap, has been developed to determine when to first execute IPColGen. The previous work of Zhao et al. (2020) identified that when the tailing-off effect is first observed the dual values tend to have stabilised. Preliminary experiments showed that a stable set of dual values improves the effectiveness of IPColGen. Additionally, IPColGen will only be executed if the optimality gap when the tailing-off effect is observed is *suitably large*, i.e. only for difficult instances. Computational experiments will be performed to identify the optimality gap level that can be regarded as *suitably large*.

In this paper, tailing-off is identified using a threshold based on the relative difference between the average LP objective value from the previous 5 iterations and the current LP objective value. If the relative difference is less than 0.01%, then the dual values have stabilised sufficiently for the execution of IPColGen. To avoid the noise in the dual solutions during the early stages of pricing, at least $|K|$ pricing iterations must be performed before the first call to IPColGen at the root node. For all subsequent nodes, at least $|K| \times 0.1$ pricing iterations must be performed before IPColGen is called.

A number of additional execution conditions must be met before IPColGen will be called at a node. These include:

- A primal feasible solution must be available prior to the execution of IPColGen. For all subsequent calls to IPColGen, a new incumbent solution must have been found since the previous call, except at the root node.

- In the branch-and-price tree, a depth frequency of 5 is used. As such, IPColGen is only called at nodes at depths $0, 5, 10, 15, \ldots$.

- IPColGen is only called when at least m nodes have been processed after the previous call. The value of m is updated based on the success of IPColGen. If IPColGen finds

an improving integer solutions, then m is set to 0, otherwise m is set to $10(m+1)$.

Since [REP] is a sub-MIP, the computational effort—in terms of nodes—is restricted to avoid unnecessarily long solution times. An esoteric formula is used for LNS heuristics within SCIP and GCG to determine the number of nodes supplied to sub-MIPs, which depends on the parameters h_p, s, and e that are the number of nodes processed, the number of solutions found by IPColGen and the number of calls of IPColGen, respectively. This formula is given by

$$h = Bh_p(1 + 2(s+1)/(e+1)) - Se + h_0,$$

where $B \in [0,1]$ is a factor of the total number of nodes processed, $S \geq 0$ corresponds to a setup cost of IPColGen in terms of nodes and h_0 is a constant offset for the number of nodes to be processed. For IPColGen the default values are $B = 0.5$, $S = 100$ and $h_0 = 500$. If h is less than a minimum number of nodes, which is 50 by default, then there is not sufficient nodes to solve the [REP] and IPColGen is not executed.

4.1 Overview of IPColGen algorithm

The IPColGen algorithm is presented in Algorithm 2. There are three main components of the IPColGen algorithm: i) forming of the partial solution, $\hat{J}$, (lines 4 and 22), ii) computing the initial dynamic penalties, $\beta_{i0}, \forall i \in I^{c0} \cup I^{p0}$, (lines 7 and 24) and iii) generating new columns by solving [REP-CG$_{ql}$], $\forall q \in Q$, and adding them to [RMP-LP] and [REP] (lines 10–18). Additionally, updating the dynamic penalties with respect to the average repair column (line 14) is critical to the success of IPColGen. The theoretical and methodological details of each of these components are outlined in Section 3. This section presents a brief overview of the implementation details related to each of the components.

The partial solution is defined by the set $\hat{J}$ and it is formed by randomly selecting a subset of the columns in the incumbent primal solution J^{IP} to be set to zero. The size of this subset is given by a run-time parameter that specifies a percentage of variables with respect to $|Q|$ that must be fixed, by default this value is 20%. Since the variables are randomly selected, it is possible that a sufficient number of variables with non-zero solution values are found. If the fixing rate is not achieved, then IPColGen is not executed.

The most important step of the heuristic is the execution of the repair pricing scheme to generate repair columns. The repair pricing scheme is executed $|Q|$ times so that a repair column is generated for each pricing problem. The order in which the pricing problems are

Algorithm 2 The IP Column Generation Heuristic

Require: A primal feasible solution J^{IP}

Ensure: An updated primal feasible solution J^{IP}

$m \leftarrow 0$, $c \leftarrow 0$
while $m < maxIter \wedge c < maxNoImprove$ **do**
    `// Create a partial solution`
    Form a partial solution from the incumbent solution
    `// Set the complementarity weight and dynamic penalties`
    Set the weight, γ, for the dual values in the pricing problem objective
    Set the initial dynamic penalties, β_{i0}, for each row $i \in I^{c0} \cup I^{p0}$
    for $n \leftarrow 0$ to 2 **do**
        `// Perform pricing to generate repair columns`
        for $l \leftarrow 0$ to $(|Q| - 1)$ **do**
            Perform the repair pricing scheme by solving [REP-CG$_{q(l)l}$]
            Add the repair column to both [RMP-LP] and [REP]
            Resolve [RMP-LP] to update the dual multipliers
            Update dynamic penalties $\beta_{i(l+1)}$, for each row $i \in I^{c0} \cup I^{p0}$
            if no dynamic penalty is updated **then**
                break
            end if
        end for
        `// Updating the incumbent and partial solution`
        if improved primal solution is found **then**
            `// Create a partial solution`
            Form a partial solution from the new incumbent solution
            `// Reset the dynamic penalties`
            Set the initial dynamic penalties, β_{i0}, for each row $i \in I^{c0} \cup I^{p0}$
        end if
    end for
    $m \leftarrow m + 1$, $c \leftarrow c + 1$
    Solve [REP] to find new solution using the repair columns
    if improved primal solution is found **then**
        $c \leftarrow 0$
    end if
end while

solved is given by the function $q(l)$, i.e. depending on the iteration of the repair pricing scheme. In practice, the order in which the pricing problems are solved is controlled internally by GCG. Each repair column that is generated during the repair pricing scheme is added to both [RMP-

LP] and [REP]. The columns are added to [RMP-LP] so that after each round of repair pricing [RMP-LP] can be resolved and the dual multipliers updated. Additionally, after executing the repair pricing scheme in iteration l, the dynamic penalties $\beta_{i(l+1)}$ are updated based on the repair columns found. Solving [REP] with the set of repair columns enables the search for columns that combine with the partial solution in an attempt to form an improving integer feasible solution.

During the execution of the repair pricing scheme, simple heuristics, such as rounding heuristics, are also called to find updated incumbent solutions. As seen in Line 22, if the simple heuristics find an improving solution, then an intermediate update of the partial solution is triggered.

Each call to the heuristic in GCG involves multiple executions of the procedure described above. The number of executions is controlled through run time parameters that specify the number of iterations and the number of iterations since the last improving solution, $maxIter$ and $maxNoImprove$ in Algorithm 2 respectively. In the implementation, the default values for $maxIter$ and $maxNoImprove$ are set to 10 and 3 respectively. In practice, the $maxIter$ limit is very rarely reached.

4.2 Updating the dynamic penalties

The method for setting the dynamic penalties given partial solution $\hat{J}$ is described in Section 3.2. In the implementation of IPColGen, the initial dynamic penalties are set to $\beta_{i0} = \beta_0, \forall i \in I^{c0} \cup I^{p0}$, where β_0 is given by a run-time parameter. As shown in Line 14 of Algorithm 2, the dynamic penalties are updated after each pricing round. In iteration l of the repair column generation procedure, if the inequality of Theorem 2 is violated for constraint $i \in I^{c0} \cup I^{p0}$, then $\beta_{i(l+1)}$ is set to $\alpha\beta_{il}$. Conversely, if the inequality of Theorem 2 is satisfied for constraint $i \in I^{c0} \cup I^{p0}$, then $\beta_{i(l+1)}$ is set to β_{il}/α. The value of α used in the computational experiments is 3. Different values of β_0 are evaluated in the computational experiments.

5 Computational experiments

A major goal of this work is to develop an LNS heuristic that will improve the effectiveness of the branch-and-price algorithm across a broad range of instance types. The effectiveness of IPColGen will be evaluated through computational experiments performed on two different

collections of instances that contain structures that are amenable to the application of IPColGen. The first set of experiments are performed on integer programming problems with known block diagonal structures that branch-and-price has traditionally been an effective solution approach. These experiments will compare the default settings of GCG against various settings for the IPColGen heuristic. The second set of experiments will evaluate how the deployment of IPColGen improves the effectiveness of applying branch-and-price to solve general MIP problems, without known block diagonal structures, using instances collected from MIPLIB 2017 (Gleixner et al., 2021).

5.1 Computational experiments setup

The computational experiments have been performed using a development version of GCG 3.1.0 (Git hash: f59eb93), that is based upon a development version of SCIP 7.0.0 (Git hash: 0bc4dc9), with SoPlex 5.0.0 (Git hash: 6535a3c) as LP solver. A computational cluster comprised of Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz CPUs and 125GB RAM per node has been used to perform the experiments. For all test instances, a time limit of 3600 seconds has been applied.

The effectiveness of IPColGen is evaluated by comparing its use against the solving performance of default GCG. Throughout this section, *IPColGen* is used to denote the parameter settings where IPColGen is deployed within the branch-price-and-cut algorithm of GCG. The label *Default* is used to denote the experiments performed using the default settings of GCG.

5.1.1 Evaluation measures

The effectiveness of IPColGen will be primarily evaluated using the measures of optimality gap and primal integral. Additionally, for some experiments the solving time and primal gap will also be evaluated. The primal integral, first proposed by Berthold (2013), aims to better capture the effect that primal heuristics have on solving performance by taking into account the quality of solutions and the time they are found.

Let $\delta_p(t)$ and $\delta_d(t)$ denote the primal and dual bounds, respectively, at time t. Thus, the

optimality gap at time t is computed by

$$\theta(t) = \begin{cases} 0 & \text{if } |\delta_p(t)| = |\delta_d(t)| = 0, \\ \dfrac{\delta_p(t) - \delta_d(t)}{\max\{\delta_p(t), \delta_d(t)\}} & \text{if } 0 < \delta_p(t) \times \delta_d(t) < \infty, \\ 1 & \text{otherwise.} \end{cases}$$

Let T denote the run time to solve the instance or the time limit if the instance is unsolved. The *final gap* refers to the optimality gap at T, which is given by $\theta(T)$. Also, *first call gap* is the label used to denote the optimality gap at the first time IPColGen is executed.

By substituting $\delta_d(t)$ with $\hat{\delta}_p$, which is the best know primal bound, $\theta_p(t)$ is used to denote the *primal gap*. The *primal integral* is then given by

$$PI = \int_0^T \theta_p(t) \mathrm{d}t.$$

To aid the comparison of parameter settings across a large diverse test set, the results will be aggregated using a shifted geometric mean. In the figures presented throughout this section, the values displayed are the ratio of the geometric mean between the evaluated setting for IPColGen and the default setting for GCG. A value greater than 1 indicates better performance by default GCG, where a value less than 1 corresponds to a performance improvement from deploying the IPColGen heuristic.

5.1.2 Selectively deploying IPColGen

Since the IPColGen heuristic is potentially highly time-consuming, due to solving multiple LP and MIP problems, it is necessary to be discriminating in its deployment. It can be expected that instances with a large optimality gap in the root node have the most to gain from deploying the proposed heuristic, while for instances with a very small optimality gap in the root node it is likely that the heuristic will deteriorate performance. Recall from Section 4 that the first call to IPColGen is made when the tailing off effect is first observed. At this point in the method, it is possible to use the optimality gap as a proxy for instance difficulty and decide whether to run the heuristic or not. To illustrate the importance of this threshold value, all plotted results are reported as a function of this value with the label *First call gap* on the horizontal axis. When results based on the shifted geometric mean are reported, the shifted geometric mean is computed using all instances with a first call gap greater than the threshold value reported on the horizontal axis.

5.2 Instances with known block diagonal structures

The first set of experiments evaluates the effectiveness of IPColGen on instances for which Dantzig-Wolfe reformulation has traditionally been effective. All problem instances, except for those for the optimal interval scheduling problem, have been collected from strIPLib (Bastubbe et al., 2021). The following instances were used for the computational experiments.

- Bin packing problem (*BPP*): A generator for bin packing problem instance, BPPGEN, was introduced in Schwerin and Wäscher (1997), which is labelled BISON in strIPLib. Three different classes of test sets are available in strIPLib, grouped by difficulty. Instances are labelled with categories $N \in \{1, 2, 3, 4\}$ corresponding to the number of items. Categories 3 and 4, which have 200 and 500 items respectively, were selected from the second class of instances, since within this subset some instances were solvable in less than 3600 seconds and others were unsolvable. The master problem is a *set covering* problem. 480 instances.

- Capacitated p-median (*CPMP*): The instances were first presented by Ceselli (2003) and Ceselli and Righini (2005). The master problem is a *set partitioning* problem. 271 instances.

- Generalised assignment problem (*GAP*): The *GAP* instances available in strIPLib have been collected from the OR Library (Beasley, 1990). For each instance, a maximisation and minimisation problem has been defined. The master problem is a *set partitioning* problem. 168 instances.

- Vertex coloring (*Coloring*): Two different sets of instances have been collected for the vertex coloring problem. These come from a compilation of instances provided by Trick (2004) (101 instances) and an instance set from the OR Library (Beasley, 1990) (20 instances). The master problem is a *set covering* problem. 121 instances.

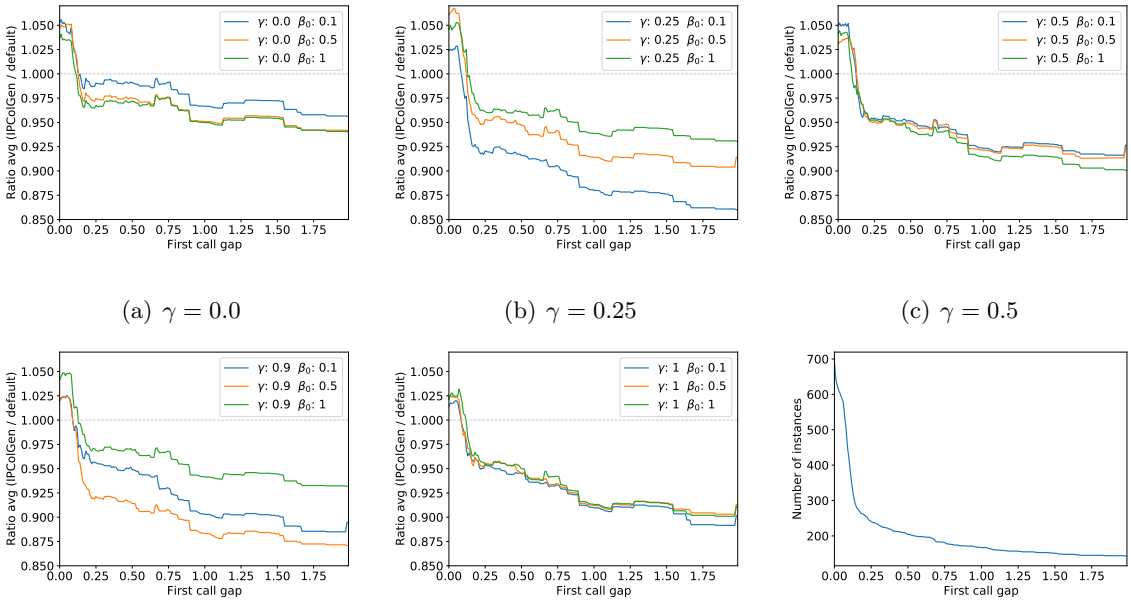
- Optimal interval scheduling problem (*OISRC*): A scheduling problem proposed by Angelelli et al. (2014) where n jobs have to be processed on m parallel machines. Additionally, multiple jobs can be processed on the same machine at one time provided resource constraints are respected. This problem tries to identify the subset of jobs that can be feasibly completed within a given time frame. Instances have been collected from `http://www.or-brescia.unibs.it`. The master problem is a *set packing* problem. 165 instances.

A total of 1205 instances form the basis of the computational experiments. From this complete test set, only the instances where IPColGen is executed at least once are considered. When IPColGen is not executed during the branch-and-price algorithm, the performance is identical to default GCG, and hence there is no performance gain or loss.

5.2.1 Evaluating different values of γ and β_0

The values of γ and β_0 are critical parameters of IPColGen that affect the search for repair columns. Figure 1 presents the ratio between the geometric means of the primal integral for *Default* and *IPColGen*. The figures present the geometric mean for all instances with a first call gap greater than the value on the horizontal axis. As explained above, only instances where IPColGen is called at least once are included in the analysis. The number of instances used to compute the geometric mean at each of the first call gap levels is displayed in Figure 1(f). Note that, as explained in Section 5.1.2, the horizontal axis is given by the *First call gap*, which is a presented in decimal form.

It can be observed that for all (γ, β_0) settings, deploying IPColGen on the complete set of instance results in a degradation in the solving performance of GCG. However, as the first



(a) $\gamma = 0.0$ (b) $\gamma = 0.25$ (c) $\gamma = 0.5$

(d) $\gamma = 0.9$ (e) $\gamma = 1.0$ (f) Number of instances

Figure 1: Relative comparison of primal integral

call gap increases—removing the easiest instances from the test set—the potential to improve the primal bound by deploying IPColGen is observed. Importantly, parity with *Default* is achieved by all (γ, β_0) settings for *IPColGen* when the first call gap is at least 13%. This indicates that for even moderately difficult instances, the use of IPColGen does not degrade the performance of GCG with respect to the primal integral.

The benefits of deploying IPColGen within GCG is best observed in Figures 1(b) and 1(d), which correspond to $\gamma = 0.25$ and $\gamma = 0.9$. A 5% improvement in the primal integral is achieved by *IPColGen* compared to *Default* on average for instances with a first call gap greater than 14% and 25% for the (γ, β_0) settings (0.25, 0.1) and (0.9, 0.5), respectively. Further, by selecting instances with a first call gap greater than 77% and 82%, at least a 10% improvement is achieved on average for the primal integral for the (0.25, 0.1) and (0.9, 0.5) settings of *IPColGen*, respectively. These results represent a significant improvement of the performance of default GCG given the diversity of the test instances.

The ability of IPColGen to find high quality solutions is highlighted by the geometric mean of the final gaps, presented in Figure 2. The most important result from Figure 2 is that all (γ, β_0) settings achieve an average improvement in the final gap, regardless of the first call gap limit. As the first call gap limit increases, the magnitude of this improvement increases. Comparing Figures 1 and 2, for all instances with a first call gap greater than 13%, *IPColGen* not only achieves a better final gap, but also achieves the corresponding primal bound improvements faster than *Default*.

The comparison between IPColGen and default GCG in regards to solving time is presented in Figure 3. Across the full test set, the number of instances solved to optimality using default GCG is 470, where for *IPColGen* using (γ, β_0) settings (0.25, 0.1) and (0.9, 0.5) this number increases to 473 and 472 respectively. In Figure 3 only the instances that are solved to optimality by both *IPColGen* and *Default* are considered. It must be noted that these are the easiest instances of the test set. Hence, these are not the target instances of IPColGen, but the solving time results are included for completeness. The number of instances used to compute the geometric mean of solving time for each first call gap limit is displayed in Figure 3(f).

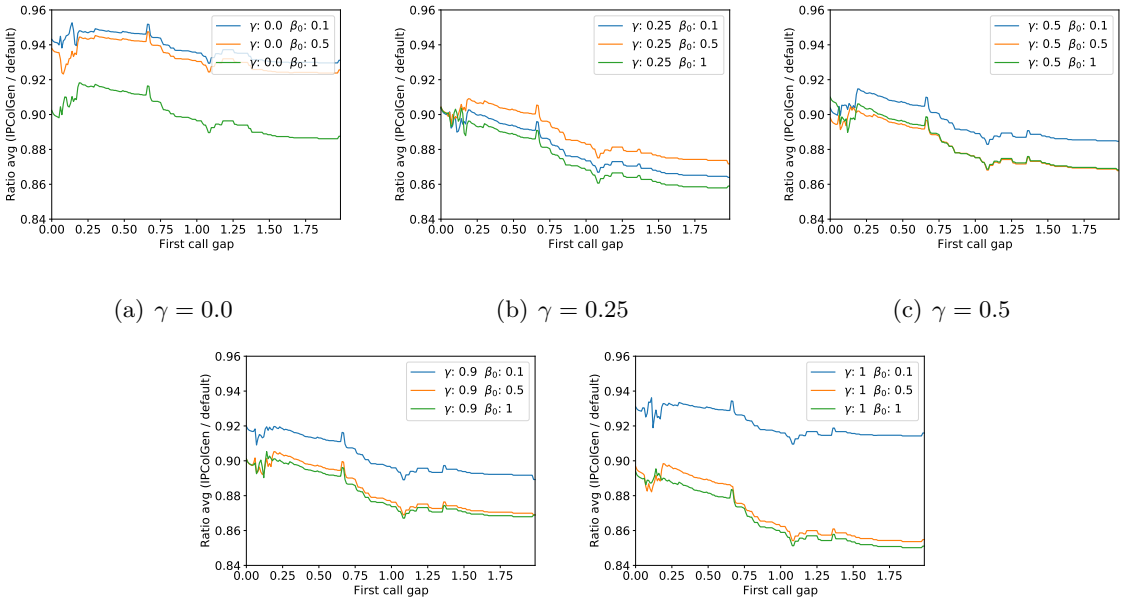
The high computational cost of IPColGen is evident from Figure 3, where the average solving time is greater for *IPColGen* compared to *Default* for many of the evaluated settings. However, for the best performing settings identified from Figures 1 and 2, given by the (γ, β_0)

pairs (0.25, 0.1) and (0.9, 0.5), as the first call gap increases the difference in the solving time decreases. In particular, for $(\gamma, \beta_0) = (0.25, 0.1)$, the average solving time of *IPColGen* equals or betters that of *Default* when the first call gap limit is at least 19%. This shows that as the difficulty of the instances increases, IPColGen is more capable of achieving an improvement in the solving time of GCG.

5.2.2 Comparison against the restricted master heuristic

The restricted master heuristic, as described by Lübbecke and Puchert (2012), is a heuristic that performs branch-and-bound to find integer feasible solutions by using only the columns generated while solving [RMP-LP]. Since IPColGen generates more—potentially better—columns at the root node, deploying the restricted master heuristic aims to evaluate whether these columns are capable of forming higher quality integer feasible solutions compared to *Default*. The relative difference of the average primal gap and primal integral between *IPColGen* and *Default* is presented in Figure 4. The parameter settings for *IPColGen* are $(\gamma, \beta_0) \in \{(0.25, 0.1), (0.9, 0.5)\}$.

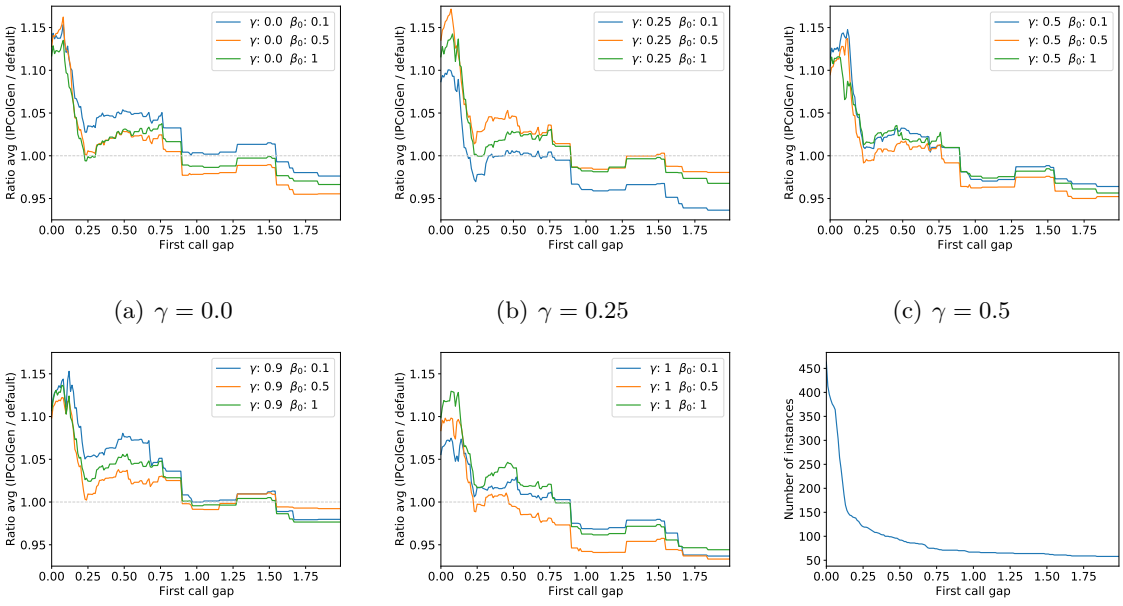
Figure 4(a) shows that deploying IPColGen significantly improves the ability of the re-



(a) $\gamma = 0.0$ (b) $\gamma = 0.25$ (c) $\gamma = 0.5$

(d) $\gamma = 0.9$ (e) $\gamma = 1.0$

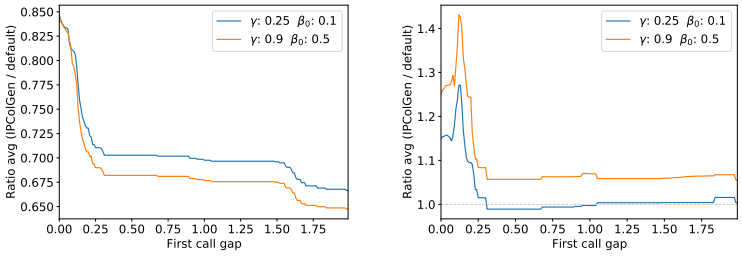
Figure 2: Relative comparison of the final gap.



(a) $\gamma = 0.0$ (b) $\gamma = 0.25$ (c) $\gamma = 0.5$

(d) $\gamma = 0.9$ (e) $\gamma = 1.0$ (f) Number of instances

Figure 3: Relative comparison of the solving time (only instances solved in 3600 seconds).



(a) Primal Gap (b) Primal Integral

Figure 4: Relative comparison of the primal gap and primal integral when using the restricted master heuristic.

stricted master heuristic to find high quality feasible solutions compared to *Default*. On the complete set of instances, *IPColGen* achieves 15% improvement in the primal gap. When considering all instances with a first call gap greater than 25%, the improvement in the primal gap is at least 29%. This result highlights the potential of the repair pricing scheme to generate columns that are capable of forming high quality feasible solutions.

This improvement in the primal gap comes at a cost in regards to the solving time. As seen in Figure 4(b), the primal integral for *IPColGen* tends to be greater than that achieved

by *Default*. This is due to the additional effort invested by *IPColGen* to generate columns that will form high quality integer solutions. Considering the results for the primal gap, presented in Figure 4(a), the increase in the computational effort can be justified given the significant improvement in solution quality that is achieved.

5.3 MIPLIB 2017 instances

To evaluate the broad applicability of IPColGen, the automatic structure detection of GCG is used to apply Dantzig-Wolfe reformulation to instances from the MIPLIB 2017 Collection. Using the MIPLIBing package (Serra and O'Neil, 2020), instances with the tags *decomposition*, *set_covering*, *set_packing* and *set_partitioning* were selected from the MIPLIB 2017 Collection. These tags were used to select instances that are most likely to have a decomposable structure and comprise constraint types necessary for the application of IPColGen. In addition, all instances for which a primal feasible solution is yet to be found, i.e. with the tag *no_solution*, were discarded from the test set. These latter instances were discarded because without a known primal bound it is not possible to compute the primal gap and, subsequently, the primal integral. This selection of the MIPLIB 2017 Collection results in a test set comprising of 190 instances. The list of instances used in the computational experiments is provided in Appendix B

In order to trigger performance variability and diversify the results, 5 different random seeds are used to permute the test instances. Each different random seed will result in a different ordering of the rows and columns when the test instance is read into GCG. In the computational results, each instance with a different random seed is considered to be a separate instance. Thus, the computational experiments have been performed using 950 instances.

Experiments have been performed using the default structure detection settings in combination with *Default* and *IPColGen*. The values for γ and β_0 are set to 0.25 and 0.1, respectively, which are the best performing settings from the previous experiments.

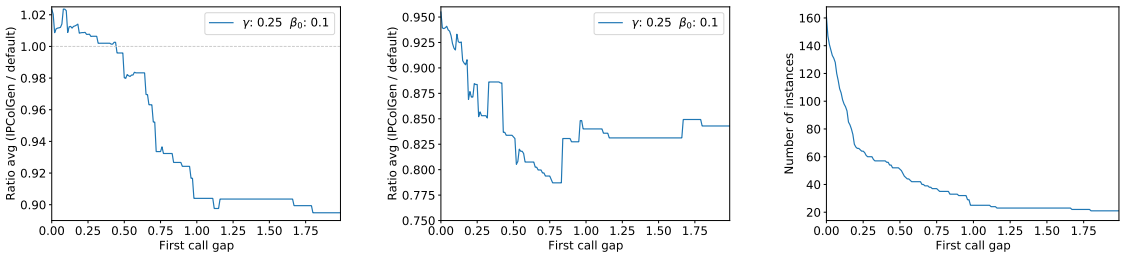
The results evaluating the improvement in the primal integral and primal gap are presented in Figure 5. Interestingly, these results show very similar behaviour to that observed in Section 5.2.1. Specifically, for the primal integral when the instances with a small first call gap are removed, i.e. the easy instances, *IPColGen* achieves an improvement over *Default*. It can be observed in Figure 5(a) that parity in the primal integral between *Default* and *IPColGen* is achieved when the first call gap is 45%, which covers more than 100 out of the 319 instances

where IPColGen is deployed. Considering the primal gap, the results presented in Figure 5(b) highlight the great ability of IPColGen to find high quality solutions. The minimum average improvement in the primal bound is 4.5% and the highest reported average improvement is 21.3%. The ability of *IPColGen* to find higher quality solutions on average across the complete test set is a valuable result for the IPColGen heuristic.

The comparison of the solving time between *Default* and *IPColGen* is presented in Figure 6. In the presented results, only the instances that were solved in 3600 seconds were used to compute the geometric means. The number of instances used to compute the geometric means is displayed in Figure 6(b). Comparing Figures 5(a) and 6(a), very similar behaviour is observed in improvement in the primal integral and solving time, respectively, when deploying IPColGen. In particular, parity is achieved for the solving time at 41%, compared to 45% for the primal integral. The reduction in the solving time when deploying IPColGen suggests that much of the improvement in the primal integral is related to the early detection of high quality integer solutions—not just improved bounds. As such, as the overall solving performance of GCG improves, IPColGen will continue to remain relevant since it is capable to finding higher quality solutions more quickly than the existing heuristics. Considering the results for the primal integral, primal gap and solving time, there is a clear benefit from deploying IPColGen within branch-price-and-cut algorithms.

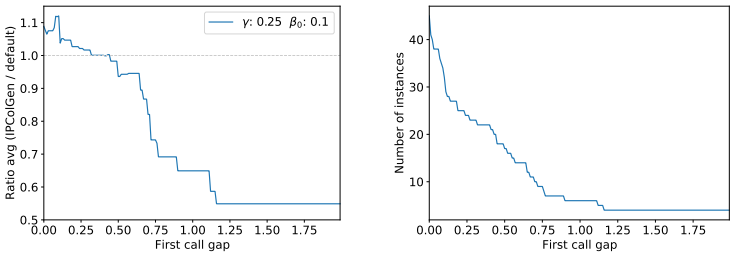
6 Conclusions

Although there are many efficient heuristics for finding high-quality primal feasible solutions to mixed-integer programs, it is inherently challenging to extend them to the branch-and-



(a) Primal Integral (b) Primal Gap (c) Number of instances

Figure 5: Relative comparison of the primal integral and primal gap.



(a) Ratio of geometric mean (b) Number of instances

Figure 6: Relative comparison of the solving time (only instances solved in 3600 seconds).

price context. The main reason is that generic heuristics are typically designed for a compact formulation where all variables are available, while in a branch-and-price setting useful variables need to be generated. By design, the generation of new variables within branch-and-price is traditionally only done for an LP relaxation via the use of column generation that relies on LP reduced costs of the variables. For this reason, the generated variables are not created with a search for high-quality primal feasible integer solutions in mind.

The contribution in our paper is IPColGen—a generic heuristic approach that can be applied when the master problem comprises set covering, set packing, or set partitioning constraints. The proposed heuristic approach is capable of being deployed as a stand-alone procedure or within a generic branch-price-and-cut solver. We handle the challenge of generating columns that have a high potential to be used in a high-quality primal feasible integer solution by generating a set of columns in each round of the repair pricing scheme such that these columns, in an average sense, have profitable properties. The question of which properties to aim for was first addressed in Zhao et al. (2020), where they were derived from the optimality conditions introduced in Larsson and Patriksson (2006). Our paper is the first to present a computationally efficient method based on these properties

IPColGen is implemented within the generic branch-price-and-cut solver GCG. The integration of IPColGen within GCG highlights the many challenges and trade-offs in the effective use of computational effort. The broad potential of IPColGen is demonstrated through an extensive computational study involving test sets with and without known block diagonal structures. On instances with large optimality gaps, significant reductions are observed in the primal integral and the final optimality and primal gaps. In comparison to the restricted master heuristic, a 29% reduction in the final primal gap is observed for instances with a first call optimality

gap greater than 25%. Finally, the deployment of IPColGen when solving general MIPLIB 2017 instances is shown to improve the performance of GCG on average. The computational results highlight the significant potential to improve the effectiveness of GCG on a broad range of instances.

The design of the repair pricing problem and the choice of parameter values in its objective are both important for the performance of IPColGen. Even if some tuning and design choices have been done for this paper, we believe that improvements of IPColGen can be made through further studies. Of particular interest is the choice of dual solution. In this paper, we have simply used the dual solution obtained when solving [RMP-LP] since it was readily available and gave good performance. From a theoretical perspective—and in contrast to the case of column generation for linear programs—it is not required to use an LP-optimal solution. Instead, one interesting option is to use a dual solution for a basic solution in the extreme point for the current primal feasible solution, since it might carry relevant information for solving the integer program. Furthermore, and despite which extreme point the dual solution belongs to, it is of interest to study which of the dual solutions for an extreme point to use, since [MP-LP] is typically expected to be highly degenerate.

Acknowledgments

The work of Elina Rönnberg is supported by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. We thank the reviewers for constructive comments that helped improve the manuscript.

A Illustration of the pricing scheme

The example is derived to illustrate the theoretical aspects of the method design as introduced in Section 3.2. The figures do not originate from a real instance. Instead a small example that covers interesting cases treated in the lemmas and theorems of Section 3.2 is created just for illustration purposes. The costs of the columns are not considered.

Let [MP] have a block diagonal structure such that there are six pricing problems with two identical pairs of each of them. This means that $Q = \{1, 2, 3\}$ and that the set of pricing problems is partitioned into subsets such that $|K_q| = 2$, $q \in Q$, hold. The introduced [MP] has nine constraints, six of covering type and three of packing type, hence $|I^{\mathrm{c}}| = 6$ and $|I^{\mathrm{p}}| = 3$.

Let the partial solution $\hat{J}$ consist of three columns such that $|\hat{J}_1| = 1$, $|\hat{J}_2| = 0$, and $|\hat{J}_3| = 2$ hold. Then, a solution to [REP] will consist of one new column from pricing problem $q = 1$, two new columns from $q = 2$ and none from $q = 3$. Note that even if the set J^{R^*} is unknown, the number of columns for each $q \in Q$ to include in a solution to [REP] is always known.

Table 1 provides information about the columns in $\hat{J}$ and introduces an ideal set of columns for [REP] to serve as reference. Such set of columns is never known when applying the repair pricing scheme, but it is included here for comparison as the existence of such set is used in the theoretical derivations.

Table 1: A [REP] and its ideal set of columns for $j \in \hat{J}_q$, $q \in Q$

	Partial solution			Set of ideal columns					Constraint type	
	$\hat{J}$			J^{R^*}			a^*	$a^*/\lvert J^{\mathrm{R}^*}\rvert$	[MP]	[REP]
j =	1	2	3	1	2	3				
q =	1	3	3	1	2	2				
	0	0	0	0	1	0	1	1/3	I^{c}	$\hat{I}^{\mathrm{c0}}$
	0	0	0	0	0	1	1	1/3	I^{c}	$\hat{I}^{\mathrm{c0}}$
	0	0	0	1	1	0	2	2/3	I^{c}	$\hat{I}^{\mathrm{c0}}$
	1	0	0	1	0	0	1	1/3	I^{c}	$\hat{I}^{\mathrm{c1}}$
	0	1	0	0	0	0	0	0/3	I^{c}	$\hat{I}^{\mathrm{c1}}$
	0	0	1	1	0	1	2	2/3	I^{c}	$\hat{I}^{\mathrm{c1}}$
	0	0	0	1	0	0	1	1/3	I^{p}	$\hat{I}^{\mathrm{p0}}$
	0	0	0	0	0	0	0	0/3	I^{p}	$\hat{I}^{\mathrm{p0}}$
	0	0	1	0	0	0	0	0/3	I^{p}	$\hat{I}^{\mathrm{p1}}$

Let the repair pricing scheme be implemented such that it, in turn for each $q \in Q$, generates $|K_q| - |\hat{J}_q|$ columns. This is repeated 3 times to obtain 9 iterations. After these iterations, the set L_q contains the columns from iterations $l = 0, 3, 6$, $l = 1, 4, 7$, and $l = 2, 5, 8$ for $q = 1$, $q = 2$, and $q = 3$, respectively. The generated columns are given in Table 2.

According to Theorem 1, a column $j \in J^{\mathrm{R}}$ can be included in a solution to [REP] only if it holds that $a_{ij} = 0$ for $i \in \hat{I}^{\mathrm{p1}}$. Since this is the case for the columns in Table 2, all of them

Table 2: Columns generated in iterations $l = 0, \ldots, 8$ for $j \in \hat{J}_q$, $q \in Q$

	Generated columns									Constraint type
l=	0	1	2	3	4	5	6	7	8	
j=	1	2	3	1	2	3	1	2	3	
q=	1	2	2	1	2	2	1	2	2	
	1	1	0	0	1	0	0	0	0	$\hat{I}^{\mathrm{c0}}$
	0	0	1	0	0	1	0	1	0	$\hat{I}^{\mathrm{c0}}$
	0	0	0	1	0	0	1	0	1	$\hat{I}^{\mathrm{c0}}$
	0	0	0	0	0	1	1	0	0	$\hat{I}^{\mathrm{c1}}$
	0	0	0	0	0	1	0	0	1	$\hat{I}^{\mathrm{c1}}$
	0	1	0	1	1	0	0	1	0	$\hat{I}^{\mathrm{c1}}$
	0	0	0	0	1	1	1	0	0	$\hat{I}^{\mathrm{p0}}$
	1	0	1	1	0	0	0	0	0	$\hat{I}^{\mathrm{p0}}$
	0	0	0	0	0	0	0	0	0	$\hat{I}^{\mathrm{p1}}$

are acceptable in this respect and the penalty M that has been used in [REP-CG$_{ql}$] appears to have been sufficient. There is no requirement on $a_{ij} = 0$, $i \in \hat{I}^{\mathrm{c1}}$, for a column $j \in J^{\mathrm{R}}$ to be included in [REP], so any value is acceptable for these coefficients.

To further analyse the behaviour of the repair pricing scheme, Tables 3, 4, and 5 provide information about the progress of the scheme with an interval of $|Q|$ iterations. In these tables, $\hat{a}_{jl}$ is the average column for the set $\hat{L}_{jl}$ and the notation

$$\mathrm{LHS}_l = \sum_{j \in J^{\mathrm{R}^*}} \sum_{j' \in \hat{L}_{jl}} a_{ij'} \quad \text{and} \quad \mathrm{RHS}_l = \frac{1}{|J^{\mathrm{R}^*}|} \sum_{j \in J^{\mathrm{R}^*}} |\hat{L}_{jl}|$$

is used for the left-hand-side and the right-hand-side, respectively, of the inequality in Theorem 2. All values, except those for a^* that are included for reference, are known from the repair pricing scheme.

After iteration $l = 2$, the inequality in Theorem 2 is violated in the third and the fifth penalty rows in Table 3, and hence the penalties for the corresponding constraints are increased. After iteration $l = 5$, the inequality is still violated for the third and the fifth rows (see Table 4) and

Table 3: Status after iteration $l = 2$

Average contribution					Status in Theorem 2				Penalty
a^*	$\hat{a}_{12}$	$\hat{a}_{22}$	$\hat{a}_{32}$	$\sum_{j \in J^{R*}} \hat{a}_{j2}$	LHS_2		RHS_2		
1	1	1	0	2	2	$\geq$	1	$i \in \hat{I}^{c0}$	$\searrow$
1	0	0	1	1	1	$\geq$	1	$i \in \hat{I}^{c0}$	$\searrow$
2	0	0	0	0	0	$\geq$	1	$i \in \hat{I}^{c0}$	$\nearrow$
1	0	0	0	0	0	$\leq$	1	$i \in \hat{I}^{p0}$	$\searrow$
0	1	0	1	2	2	$\leq$	1	$i \in \hat{I}^{p0}$	$\nearrow$

Table 4: Status after iteration $l = 5$

Average contribution					Status in Theorem 2				Penalty
a^*	$\hat{a}_{15}$	$\hat{a}_{25}$	$\hat{a}_{35}$	$\sum_{j \in J^{R*}} \hat{a}_{j5}$	LHS_5		RHS_5		
1	1/2	1	0	3/2	3	$\geq$	2,	$i \in \hat{I}^{c0}$	$\searrow$
1	0	0	1	1	2	$\geq$	2,	$i \in \hat{I}^{c0}$	$\searrow$
2	1/2	0	0	1/2	1	$\geq$	2,	$i \in \hat{I}^{c0}$	$\nearrow$
1	0	1/2	1/2	1	2	$\leq$	2,	$i \in \hat{I}^{p0}$	$\searrow$
0	1	0	1/2	3/2	3	$\leq$	2,	$i \in \hat{I}^{p0}$	$\nearrow$

the penalty is further increased. After iteration $l = 8$ (see Table 5), all rows comply with the inequality in Theorem 2.

Since the columns generated during these nine iterations comply with the conditions in Theorem 1 and Theorem 2, it is likely that some of them are good candidates for inclusion in a high-quality solution to [REP]. One possible solution is to use the columns from iterations $l = 0, 7, 8$. Note, however, that the ideal columns have not been generated. There is never a guarantee that the repair pricing scheme will generate the ideal columns.

Table 5: Status after iteration $l = 8$

Average contribution					Status in Theorem 2				Penalty
a^*	$\hat{a}_{18}$	$\hat{a}_{28}$	$\hat{a}_{38}$	$\sum_{j \in J^{\mathrm{R}*}} \hat{a}_{j8}$	LHS_8		RHS_8		
1	1/3	2/3	0	1	3	$\geq$	3,	$i \in \hat{I}^{\mathrm{c0}}$	$\searrow$
1	0	1/3	2/3	1	3	$\geq$	3,	$i \in \hat{I}^{\mathrm{c0}}$	$\searrow$
2	2/3	0	1/3	1	3	$\geq$	3,	$i \in \hat{I}^{\mathrm{c0}}$	$\searrow$
1	1/3	1/3	1/3	1	3	$\leq$	3,	$i \in \hat{I}^{\mathrm{p0}}$	$\searrow$
0	2/3	0	1/3	1	3	$\leq$	3,	$i \in \hat{I}^{\mathrm{p0}}$	$\searrow$

B MIPLIB 2017 Instances

The instances from the MIPLIB 2017 Collection that were chosen for the computational experiments are given in Table 6. The MIPLIBing package (Serra and O'Neil, 2020) was used for selecting and downloading the instances from the MIPLIB 2017 Collection.

Table 6: Selected MIPLIB 2017 instances.

30_70_45_05_100	graph20-80-1rand	neos-5261882-treska
30_70_45_095_100	graph40-20-1rand	neos-5266653-tugela
30_70_45_095_98	gus-sch	neos-555001
ab51-40-100	h50x2450	neos-555884
ab67-40-100	h80x6320	neos-574665
ab69-40-100	h80x6320d	neos-595904
ab71-20-100	lotsize	neos-824661
ab72-40-100	maritime-jg3d9	neos-826224
academictimetablesmall	mcsched	neos-826650
aflow30a	mitre	neos-827175
aflow40b	mkc	neos-876808
allcolor10	mtest4ma	neos-913984
allcolor58	n2seq36f	neos-914441

Continued on next page

Table 6 – *Continued from previous page*

app1-2	n3div36	neos-932721
app2-1	nb10tb	neos-935769
app2-2	neos-1140050	neos-941313
atm20-100	neos-1337307	neos-953928
bab1	neos-1354092	neos18
bab2	neos-1420546	neos2
bab3	neos-1420790	neos4
bab5	neos-1456979	neos9
beavma	neos-1516309	net12
blp-ar98	neos-1582420	nexp-50-20-4-2
blp-ic97	neos-1593097	no-ip-64999
blp-ic98	neos-1599274	no-ip-65059
blp-ir98	neos-2294525-abba	ns1111636
bmocbd	neos-2328163-agri	ns1631475
bmocbd2	neos-2624317-amur	ns2122698
bmoipr2	neos-2626858-aoos	ns2124243
brazil3	neos-2652786-brda	nursesched-medium-hint03
cap6000	neos-2656603-coxs	nursesched-medium04
circ10-3	neos-2657525-crna	nursesched-sprint-late03
comp07-2idx	neos-2669500-cust	physicianssched3-3
comp08-2idx	neos-2746589-doon	physicianssched5-3
comp12-2idx	neos-3045796-mogo	physicianssched6-1
comp16-3idx	neos-3048764-nadi	physicianssched6-2
comp21-2idx	neos-3075395-nile	piperout-d20
control20-5-10-5	neos-3083784-nive	piperout-d27
control30-3-2-3	neos-3116779-oban	qnet1_o
control30-5-10-4	neos-3118745-obra	rococoB10-011000
cost266-UUE	neos-3148108-pahi	rococoC11-010100
cvs08r139-94	neos-3209519-ruhr	rococoC11-011100
cvs16r106-72	neos-3218348-suir	rococoC12-010001

Continued on next page

Table 6 – *Continued from previous page*

cvs16r128-89	neos-3355120-tarago	rout
cvs16r70-62	neos-3581454-haast	sct32
cvs16r89-60	neos-3695882-vesdre	shiftreg1-4
d20200	neos-3696678-lyvia	shiftreg2-7
dcmulti	neos-3734794-moppy	sing11
decomp1	neos-3988577-wolgan	sing326
decomp2	neos-4295773-pissa	sing44
dws008-01	neos-4300652-rahue	sing5
dws008-03	neos-4322846-ryton	sp97ic
dws012-01	neos-4360552-sangro	sp98ic
fastxgemm-n2r6s0t2	neos-4408804-prosna	sp98ir
fastxgemm-n2r7s4t1	neos-4531126-vouga	stp3d
fastxgemm-n3r21s3t6	neos-4533806-waima	supportcase29
fastxgemm-n3r22s4t6	neos-4555749-wards	supportcase31
fastxgemm-n3r23s5t6	neos-4736745-arroux	supportcase40
fhnw-binschedule0	neos-4738912-atrato	traininstance2
fhnw-binschedule2	neos-4954274-beardy	umts
fiball	neos-5041756-cobark	vpphard
fiber	neos-5188808-nattai	vpphard2
germanrr	neos-525149	
graph20-20-1rand	neos-5260764-orauea	

References

F. Alvelos, A. de Sousa, and D. Santos. SearchCol: Metaheuristic search by column generation. In *Hybrid Metaheuristics: 7th International Workshop, HM 2010, Vienna, Austria, October 1-2, 2010, Proceedings*, volume 6373 of *Lecture Notes in Computer Science*, pages 190–205. 2010.

F. Alvelos, E. Silva, and J. M. V. de Carvalho. A hybrid heuristic based on column generation for two- and three-stage bin packing problems. In *Computational Science and Its Applications – ICCSA 2014: 14th International Conference, Guimarães, Portugal, June 30 – July 3, 2014, Proceedings, Part II*, pages 211–226. 2014.

Enrico Angelelli, Nicola Bianchessi, and Carlo Filippi. Optimal interval scheduling with a resource constraint. *Computers & Operations Research*, 51:268 – 281, 2014. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2014.06.002. URL `http://www.sciencedirect.com/science/article/pii/S0305054814001609`.

R. Baldacci, L. Bodin, and A. Mingozzi. The multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem. *Computers and Operations Research*, 33: 2667–2702, 2006.

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

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46 (3):316–329, 1998.

Michael Bastubbe, Alexander Helber, Lukas Kirchhart, Marco Lübbecke, Niklas Rieken, and Jonas Witt. strIPlib: Structured Integer Programming Library, 2021. URL `https://striplib.or.rwth-aachen.de/`.

M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, New York, second edition, 1993.

J. E. Beasley. OR-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990. ISSN 01605682, 14769360. URL http://www.jstor.org/stable/2582903.

M. Bergner, A. Caprara, A. Ceselli, F. Furini, M.E. Lübbecke, E. Malaguti, and E. Traversi. Automatic Dantzig-Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1–2):391–424, 2015. doi: 10.1007/s10107-014-0761-5. URL http://www.or.rwth-aachen.de/research/publications/s10107-014-0761-5.pdf.

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41 (6):611–614, 2013. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2013.08.007. URL http://www.sciencedirect.com/science/article/pii/S0167637713001181.

Timo Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, 2014. URL http://www.zib.de/berthold/Berthold2014.pdf.

D. Bredström, K. Jörnsten, M. Rönnqvist, and M. Bouchard. Searching for optimal integer solutions to set partitioning problems using column generation. *International Transactions in Operational Research*, 21(2):177–197, 2014.

Alberto Ceselli. Two exact algorithms for the capacitated p-median problem. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4):319–340, 2003.

Alberto Ceselli and Giovanni Righini. A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142, 2005. doi: https://doi.org/10.1002/net.20059. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20059.

Coluna. Coluna.jl. See https://github.com/atoptima/Coluna.jl, 2021. Last accessed: 09-07-2021.

J. Desrosiers and M. Lübbecke. Branch-price-and-cut algorithms. In J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, and J.C. Smith, editors, *Encyclopedia of Operations Research and Management Science*, Hoboken, NJ, Jan 2011. John Wiley & Sons.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003. ISSN 0025-5610. URL http://dx.doi.org/10.1007/s10107-003-0395-5.

M.V. Galati, T.K. Ralphs, and J. Wang. Computational experience with generic decomposition using the DIP framework. In *Proceedings of RAMP 2012*, 2012.

Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical Report 20-10, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, 2020.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13:433–490, 2021.

E. L. Johnson. Modeling and strong linear programs for mixed integer programming. In Stein W. Wallace, editor, *Algorithms and model formulations in mathematical programming*, pages 1–43. Springer, Berlin Heidelberg, 1989.

C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010.

T. Larsson and M. Patriksson. Global optimality conditions for discrete and nonconvex optimization-with applications to Lagrangian heuristics and column generation. *Operations Research*, 54(3):436–453, 2006.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(3):1007–1023, 2005.

Marco Lübbecke and Christian Puchert. Primal heuristics for branch-and-price algorithms. In *Operations Research Proceedings 2011*, pages 65–70. Springer, 2012.

Pierre Pesneau, Ruslan Sadykov, and François Vanderbeck. Feasibility pump heuristics for column generation approaches. In *International Symposium on Experimental Algorithms*, pages 332–343. Springer, 2012.

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

David Pisinger and Stefan Ropke. *Large Neighborhood Search*, pages 399–419. Springer US, Boston, MA, 2010.

E. Rönnberg and T. Larsson. Column generation in the integral simplex method. *European Journal of Operational Research*, 192(1):333–342, 2009.

E. Rönnberg and T. Larsson. All-integer column generation for set partitioning: Basic principles and extensions. *European Journal of Operational Research*, 233(3):529–538, 2014.

Elina Rönnberg and Torbjörn Larsson. An integer optimality condition for column generation on zero–one linear programs. *Discrete Optimization*, 31:79–92, 2019. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2018.09.001. URL `https://www.sciencedirect.com/science/article/pii/S1572528616301050`.

R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal heuristics for branch and price: The assets of diving methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

Ruslan Sadykov and François Vanderbeck. BaPCod - a generic branch-and-price code. Technical report, Inria Bordeaux Sud-Ouest, November 2021. URL `https://hal.inria.fr/hal-03340548`.

Petra Schwerin and Gerhard Wäscher. The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4(5):377–389, 1997. ISSN 0969-6016. doi: https://doi.org/10.1016/S0969-6016(97)00025-7. URL `https://www.sciencedirect.com/science/article/pii/S0969601697000257`.

Thiago Serra and Ryan J O'Neil. MIPLIBing: Seamless benchmarking of mathematical optimization problems and metadata extensions. *SN Operations Research Forum*, 1(3):1–6, 2020.

Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming - CP98. CP 1998. Lecture Notes in Computers Science*, volume 1520, pages 417–431, 1998.

D. J. Sweeney and R. A. Murphy. A method of decomposition for integer programs. *Operations Research*, 27(6):1128–1141, 1979.

Adil Tahir, Guy Desaulniers, and Issmail El Hallaoui. Integral column generation for the set partitioning problem. *EURO Journal on Transportation and Logistics*, 8(5):713–744, 2019.

G. L. Thompson. An integral simplex algorithm for solving combinatorial optimization problems. *Computational Optimization and Application*, 22(3):351–367, 2002.

Michael Trick. Computational series: Graph coloring and its generalizations. See `https://mat.tepper.cmu.edu/COLOR04/`, 2004. Last accessed 01-06-2021.

V. A. Trubin. On a method of solution of integer linear programming problems of a special kind. Translated by V. Hall. *Soviet Mathematics Doklady*, 10:1544–1546, 1969.

F. Vanderbeck. *Decomposition and column generation for integer programs.* PhD thesis, UCL-Université Catholique de Louvain, 1994.

A. Zaghrouti, I. El Hallaoui, and F. Soumis. Integral simplex using decomposition for the set partitioning problem. *Operations Research*, 62(2):435–449, 2014.

Y. Zhao, T. Larsson, and E. Rönnberg. An integer programming column generation principle for heuristic search methods. *International Transactions in Operational Research*, 27(1): 665–695, 2020.