Les Cahiers du GERAD ISSN: 0711–2440

Integral column generation for set partitioning problems with side constraints

A. Tahir,
G. Desaulniers, I. El Hallaoui

G–2019–85

November 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Integral column generation for set partitioning problems with side constraints

Adil Tahir [a,b]

Guy Desaulniers [a,b]

Issmail El Hallaoui [a,b]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

adil.tahir@gerad.ca
guy.desaulniers@gerad.ca
issmail.elhallaoui@gerad.ca

November 2019
Les Cahiers du GERAD
G–2019–85

Abstract: The integral column generation algorithm (ICG) was recently introduced to solve set partitioning problems involving a very large number of variables. This primal algorithm generates a sequence of integer solutions with decreasing costs, leading to an optimal or near-optimal solution. ICG combines the well-known column generation algorithm and a primal algorithm called the integral simplex using decomposition algorithm (ISUD). In this paper, we develop a generalized version of ICG, denoted I^2CG, that can solve efficiently large-scale set partitioning problems with side constraints. This new algorithm can handle the side constraints in the reduced problem of ISUD, in its complementary problem or in both components. Computational experiments on instances of the airline crew pairing problem involving up to 1761 constraints show that the latter strategy is the most efficient one and that I^2CG significantly outperforms two popular column generation heuristics, namely, a restricted master heuristic and a diving heuristic. For the largest tested instance, I^2CG can produce in less than one hour of computational time more than 500 integer solutions leading to an optimal or near-optimal solution.

Keywords: Discrete optimization, integral column generation, integral simplex using decomposition, aircrew scheduling

1 Introduction

The set partitioning problem (SPP) is one of the fundamental models used, in particular, for vehicle routing and crew scheduling. In a generic way, these problems can be stated as finding a least-cost set of paths in a suitable network such that they cover a set of tasks exactly once each. A task can be a customer to visit or a flight to cover, whereas a path represents a vehicle route or a bus driver/aircrew schedule. For large-scale problem instances, these paths are generated by a column generation (CG) technique because it is impossible to enumerate all of them a priori. Side constraints such as vehicle or pilot availability are often added to the SPP, yielding the SPP with side constraints (SPPSC) that we consider in this paper.

Let $\mathbf{T} = \{1, \ldots, m\}$ be the set of tasks (associated with the set partitioning constraints), $\mathbf{H}$ the set of side constraint indices, and $\mathbf{N} = \{1, \ldots, n\}$ the set of paths (columns). For each column $j \in \mathbf{N}$, define a binary variable x_j that is equal to 1 if column j with cost c_j is part of the solution and 0 otherwise. Furthermore, for each column $j \in \mathbf{N}$ and task $i \in \mathbf{T}$, let a_{ij} be a binary parameter equal to 1 if column j covers task i and 0 otherwise. These coefficients a_{ij} define the set partitioning constraint matrix $\mathbf{A} = (A_j)_{j\in\mathbf{N}} = (a_{ij})_{i\in\mathbf{T}, j\in\mathbf{N}}$. For each $j \in \mathbf{N}$, we assume that column A_j covers at least one task, i.e., there exists $i \in \mathbf{T}$ such that $a_{ij} = 1$. Finally, the coefficient q_{hj}, $j \in \mathbf{N}$, $h \in \mathbf{H}$, denotes the contribution of variable x_j to the side constraint h, whose right-hand side is b_h.

Given this notation, the SPPSC can be written as the following integer program:

$$(\mathbb{P}) \qquad \min_{x} \quad \sum_{j\in\mathbf{N}} c_j x_j \tag{1}$$

$$\text{s.t.:} \quad \sum_{j\in\mathbf{N}} a_{ij} x_j = 1, \qquad \forall\, i \in \mathbf{T} \tag{2}$$

$$\sum_{j\in\mathbf{N}} q_{hj} x_j \le b_h, \qquad \forall\, h \in \mathbf{H} \tag{3}$$

$$x_j \in \{0, 1\}, \qquad \forall\, j \in \mathbf{N}. \tag{4}$$

The objective function (1) aims at minimizing the total cost of the selected columns. The set partitioning constraints (2) ensure that each task is covered exactly once. The problem side constraints are expressed by (3). Note that we will focus only on less-than-or-equal-to side constraints but the proposed theory and methodology remain valid for equalities and greater-than-or-equal-to inequalities. Finally, binary requirements on the variables are imposed through (4). The SPP model is given by (1), (2), and (4).

In the rest of this paper, we denote by $\mathcal{F}_{\mathcal{SPPSC}}$, $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$ and $conv(\mathcal{F}_{\mathcal{SPPSC}})$ the feasible domains of the SPPSC model $\mathbb{P}$ and its linear relaxation, and the convex hull of $\mathcal{F}_{\mathcal{SPPSC}}$, respectively. Similarly, we write $\mathcal{F}_{\mathcal{SPP}}$, $\mathcal{F}_{\mathcal{SPP}^{\mathcal{LR}}}$ and $conv(\mathcal{F}_{\mathcal{SPP}})$ for the SPP, i.e., model $\mathbb{P}$ without the side constraints (3).

Because it includes the side constraints (3), model $\mathbb{P}$ does not possess an interesting property of the SPP model, called *quasi-integrality*. This property implies that, from any integer extreme point of $\mathcal{F}_{\mathcal{SPP}^{\mathcal{LR}}}$, there exists a so-called *integer* path along the edges of $\mathcal{F}_{\mathcal{SPP}^{\mathcal{LR}}}$ that visits only integer extreme points with decreasing costs and leads to an optimal solution. Consequently, the side constraints (3) add an extra difficulty to the SPP, which is already known for its high degeneracy. This difficulty mainly hinders the primal solution algorithms which aim at finding a sequence of integer solutions with decreasing costs that ends with an optimal or a quasi-optimal solution. Indeed, these algorithms can get stuck in a local minimum when the side constraints cut all integer paths between the current solution and all optimal ones.

The goal of this paper is to develop a new primal algorithm that can efficiently solve the SPPSC and must, therefore, overcome high degeneracy and the loss of the quasi-integrality property. To achieve this objective, we add artificial edges to $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$ to create new integer paths that can be traversed to reach optimal solutions. Furthermore, we generalize the integral CG algorithm (ICG)

of Tahir et al. [17] that was designed for the SPP and combines CG and the integral simplex using decomposition algorithm (ISUD [21]), a recent primal algorithm which exploits degeneracy instead of suffering from it. Because ISUD decomposes the problem into a reduced problem (RP) and a complementary problem (CP), we study different variants of this improved ICG algorithm, denoted I^{2}CG, where the side constraints are handled in the RP, the CP or in both subproblems. These I^{2}CG variants are tested on real-world instances of the airline crew pairing problem (CPP) involving up to 1761 constraints and benchmarked against two popular CG heuristics. Our computational results show that, when compared to these two heuristics, I^{2}CG can compute better-quality solutions in less than 35% of the computational time on average.

This paper is organized as follows. In Section 2, we review the literature on the most commonly used dual and primal algorithms for solving the SPP or the SPPSC before summarizing our main contributions. In Section 3, we study, via an example, the effect of the side constraints on $\mathcal{F}_{\mathcal{SPPLR}}$ and introduce a new quasi-integrality property that is exploited by the proposed algorithm. Then, in Section 4, we describe two approaches to handle the side constraints and present theoretical results to support the second one. Next, I^{2}CG and its variants are detailed in Section 5. Finally, computational results on CPP instances are reported in Section 6 before drawing conclusions in Section 7.

2 Literature review and contributions

The SPP and the SPPSC can be tackled with several solution algorithms. Letchford and Lodi [12] have divided these algorithms into three classes: dual integral, dual fractional, and primal algorithms. Dual integral algorithms maintain integrality and dual feasibility throughout the solution process and strive to reach primal feasibility. Because this class contains a single algorithm, that of Gomory [9], we focus on the other two classes below.

2.1 Dual fractional algorithms

Dual fractional algorithms relax the binary requirements but maintain at each iteration primal and dual feasibility. Cutting plane algorithms as well as branch-and-bound and branch-and-cut algorithms fall in this class. For solving SPPs or SPPSCs involving a very large number of variables, branch-and-price (BP, Barnhart et al. [3], Desaulniers et al. [5]) is one of the best-known dual fractional algorithms. BP is a branch-and-bound algorithm where the linear relaxation at each node of the search tree is solved by CG. CG is an iterative method that solves a linear program by decomposing it into a restricted master problem (RMP) and a pricing subproblem (PS). For the SPPSC, the RMP is the linear relaxation of (1)–(3) restricted to a subset of its variables, whereas the PS can be, for example, a shortest path problem with resource constraints. At each iteration of CG, the RMP is first solved by a linear programming solver to provide a primal and a dual solution $\pi = \left((\pi_t)_{t\in\mathbf{T}}, (\pi_h)_{h\in\mathbf{H}}\right) \in \mathbb{R}^{|\mathbf{T}|} \times \mathbb{R}_-^{|\mathbf{H}|}$. Then, the PS is solved by a specialized algorithm, e.g., a labeling algorithm, to find variables x_j that are not in the RMP and that have a negative reduced cost $\bar{c}_j = c_j - \sum_{i\in\mathbf{T}} \pi_i a_{ij} - \sum_{h\in\mathbf{H}} \pi_h q_{hj}$. These variables (columns) are added to the RMP which is re-optimized to start a new iteration. When no variables are generated, CG stops and the current RMP solution is optimal for the relaxed SPPSC.

To improve the effectiveness of branch-and-price algorithms, several research works have been conducted to reduce the impact of factors influencing the convergence of the standard CG (see [20]). With the same purpose, El Hallaoui et al. [8, 7] introduced the dynamic constraint aggregation algorithm (DCA) and the multi-phase DCA (MPDCA). These two algorithms reduce the impact of the degeneracy in the RMP and the number of fractional variables in the linear relaxation solutions by lowering the number of set partitioning constraints in the RMP. Bouarab et al. [4] generalized this work by combining CG and the improved primal simplex algorithm (IPS, [6]), which is effective against degeneracy for general linear programs.

2.2 Primal algorithms

Unlike dual algorithms, primal algorithms maintain the feasibility of all constraints (2)–(3), including the binary requirements. They stop when optimality is reached. In other words, these algorithms search for a decreasing sequence of integer solutions leading to an optimal solution. For the SPP, the existence of this sequence has been proven by Balas and Padberg [1, 2]. This result is based on the *quasi-integrality* property proved by Trubin [19] which indicates that the edges of $conv(\mathcal{F}_{\mathcal{SPP}})$ are also edges of $\mathcal{F}_{\mathcal{SPP}^{\mathcal{LR}}}$. This property will be discussed in detail in Section 3. Several authors have studied the SPP to develop new algorithms that search for such a sequence of integer solutions. The algorithms proposed by Rönnberg and Larsson [13, 14] and Thompson [18] explore, by enumeration, the adjacent degenerate bases associated with an extreme point. Their objective is to find a basis inducing a non-degenerate pivot that improves the solution. These highly combinatorial algorithms are not effective for large-scale SPP instances, mainly due to severe degeneracy.

In the same perspective, Zaghrouti et al. [21] proposed a new primal algorithm, called ISUD. The latter decomposes the SPP into a RP and a CP. The RP is defined as the SPP restricted to subsets of its variables and constraints (including integrality). It searches for an improved solution in the vector subspace generated by the columns of the current integer solution support. In the complementary subspace, the CP looks for descent directions qualified as *integer*.

Definition 1 *A descent direction $d \in \mathbb{R}^{|\mathbf{N}|}$ is said to be minimal with respect to a solution x^0 if and only if (i) pivoting on all the variables x_j, $j \in \{j \in \mathbf{N} | d_j > 0\}$, allows to reach an extreme point adjacent to x^0; (ii) pivots on any strict subset of these variables are degenerate. In addition, d is said to be integer if it leads to an integer solution (i.e., $x^0 + d$ is integer); otherwise, it is said to be fractional.*

Like other primal algorithms, ISUD starts with an integer solution x^0. Denote by $\mathcal{S} = supp(x^0) = \{j \in \mathbf{N} \mid x^0_j \neq 0\}$, the column index set of the solution support. By breach of terminology, we say that $\mathcal{S}$ is the solution x^0 and that columns A_j, $j \in \mathcal{S}$, are in solution x^0 or in $\mathcal{S}$. The other columns A_j, $j \in \mathbf{N} \setminus \mathcal{S}$, are either compatible or incompatible according to the following compatibility definition (Elhallaoui et al. [6]).

Definition 2 *A subset $\mathcal{U}$ of $\mathbf{N}$ is said to be compatible with $\mathcal{S}$, or simply compatible, if there exist two vectors $v \in \mathbb{R}^{|\mathcal{U}|}_+$ and $\lambda \in \mathbb{R}^{|\mathcal{S}|}$ such that $\sum_{j\in\mathcal{U}} v_j A_j = \sum_{l\in\mathcal{S}} \lambda_l A_l$. The combination of columns, possibly a singleton, $\sum_{j\in\mathcal{U}} v_j A_j$ is also qualified as compatible.*

Let $\mathbf{C}_\mathcal{S}$ be the index set of the individually compatible columns (including those in $\mathcal{S}$) and $\mathbf{I}_\mathcal{S}$ the index set of the incompatible columns. Note that $\mathbf{I}_\mathcal{S} \cap \mathbf{C}_\mathcal{S} = \emptyset$. The compatible columns are considered in the RP, whose mathematical formulation is as follows:

$$(RP1) \quad \min_x \quad \sum_{j\in\mathbf{C}_\mathcal{S}} c_j x_j \tag{5}$$

$$\text{s.t.:} \quad \sum_{j\in\mathbf{C}_\mathcal{S}} a_{ij} x_j = 1, \qquad \forall\, i \in \mathbf{T}' \tag{6}$$

$$x_j \in \{0,1\}, \qquad \forall\, j \in \mathbf{C}_\mathcal{S} \tag{7}$$

where $\mathbf{T}' \subseteq \mathbf{T}$ is the index set of a maximal subset of linearly independent rows of matrix $\mathbf{A}$ when restricted to the columns in $\mathbf{C}_\mathcal{S}$ (e.g., subset of the first rows covered by each column A_j, $j \in \mathcal{S}$). RP1 can easily be solved using a commercial mixed-integer programming solver (e.g., Cplex). Note that the columns A_j, $j \in \mathcal{S}$, forms a non-degenerate basis of the constraint matrix of RP1 and, if there exists a variable x_j, $j \in \mathbf{C}_\mathcal{S} \setminus \mathcal{S}$, with a negative reduced cost with respect to the corresponding dual solution, then pivoting this variable into the basis yields an improved solution. In this case, the sets $\mathcal{S}$ and $\mathbf{C}_\mathcal{S}$ are updated and this process is repeated as long as RP1 finds an improved solution. Otherwise, the CP is solved to find a minimal descent direction $d \in \mathbb{R}^{|\mathbf{N}|}$. Algebraically, d is integer if the combination of incompatible columns A_j, $j \in \mathcal{U}$, is compatible with $\mathcal{S}$ and composed of disjoint columns according to the following definition.

Definition 3 *Let $\mathcal{U} \subset \mathbf{N}$ be a subset of column indices. The columns A_j, $j \in \mathcal{U}$, are said to be disjoint if and only if $A_{j_1}^\top A_{j_2} = 0$, $\forall j_1, j_2 \in \mathcal{U}$, $j_1 \neq j_2$.*

Let v_j, $j \in \mathbf{I}_\mathcal{S}$, be the variables defining the weights of the incompatible columns in a linear combination $\sum_{j\in\mathbf{I}_\mathcal{S}} v_j A_j$. Let λ_l, $l \in \mathcal{S}$, be the variables defining the weights of the columns in a linear combination $\sum_{l\in\mathcal{S}} \lambda_l A_l$. Let $\mathbf{F}_\mathcal{S} = \{(j_1, j_2) \in \mathbf{I}_\mathcal{S} \times \mathbf{I}_\mathcal{S} | A_{j_1}^\top A_{j_2} \neq 0, j_1 \neq j_2\}$ be the set of all pairs of non-disjoint incompatible column indices. The CP, denoted CP1 to indicate its first version, is formulated as follows:

$$(CP1) \qquad z^{CP1} = \min_{v,\lambda} \quad \sum_{j\in\mathbf{I}_\mathcal{S}} c_j v_j - \sum_{l\in\mathcal{S}} c_l \lambda_l \tag{8}$$

$$\text{s.t.:} \quad \sum_{j\in\mathbf{I}_\mathcal{S}} a_{ij} v_j - \sum_{l\in\mathcal{S}} a_{il} \lambda_l = 0, \qquad \forall\, i \in T \tag{9}$$

$$\sum_{j\in\mathbf{I}_\mathcal{S}} \alpha_j v_j + \sum_{l\in\mathcal{S}} \beta_l \lambda_l = 1 \tag{10}$$

$$v_j \geq 0, \qquad \forall j \in \mathbf{I}_\mathcal{S} \tag{11}$$

$$v_{j_1} v_{j_2} = 0, \qquad \forall (j_1, j_2) \in \mathbf{F}_\mathcal{S}. \tag{12}$$

The objective function (8) aims at minimizing the reduced cost of the chosen linear combination of incompatible columns $\sum_{j\in\mathbf{I}_\mathcal{S}} v_j A_j$. Constraints (9) ensure that this linear combination is compatible with $\mathcal{S}$. Involving the non-negative scalars α_j, $j \in \mathbf{I}_\mathcal{S}$, and β_l, $l \in \mathcal{S}$, the normalization constraint (10) bounds the problem (otherwise, CP1 would be unbounded when an integer descent direction exists). Constraints (12) ensure that the selected incompatible columns A_j are pairwise disjoint. If $z^{CP1} < 0$, then an integer descent direction is found and the solution $\mathcal{S}$ is improved by replacing the columns A_l, $l \in \mathcal{S}$, such that $\lambda_l > 0$ with the columns A_j, $j \in \mathbf{I}_\mathcal{S}$, such that $v_j > 0$ (i.e., $d = \left(d_j\right)_{j\in\mathbf{N}}$ with $d_j = 1$ if $j \in \mathbf{I}_\mathcal{S}$ and $v_j > 0$, $d_j = -1$ if $j \in \mathcal{S}$ and $\lambda_j > 0$, and $d_j = 0$ otherwise). It should be noted that in practice, the constraints (12) are relaxed from CP1 to yield a linear program. As shown by Zaghrouti et al. [21], this relaxed CP1 often finds integer descent directions. When this is not the case, branching can be done to cut the fractional direction. In the following, we keep using CP1 to denote its relaxed version.

Alternating between solving RP1 and CP1 allows ISUD to find a decreasing sequence of integer solutions leading to an optimal solution. The computational experiments carried out by Zaghrouti et al. [21] on 90 instances of the vehicle and crew scheduling problem (VCSP) and the CPP involving up to 1600 constraints and 500,000 variables (generated a priori) have shown the effectiveness of ISUD.

However, ISUD has two major limitations. First, CP1 may find fractional descent directions. Second, ISUD can only reach at each iteration the extreme points that are adjacent to the current integer solution. To overcome the first limitation, Rosat et al. [15, 16] proposed to add cuts to CP1 to exclude fractional directions. Finding these cuts is usually time-consuming and several of them may be required to derive an integer direction. These authors also proved that the integrality of the directions found by CP1 is influenced by the weights of the variables in the normalization constraint (10).

Alternatively, Zaghrouti et al. [23] developed a variant of ISUD, called ZOOM, that zooms in the neighborhood of a fractional direction to look for an improved integer solution. To do so, it adds a subset of incompatible columns to the RP which is then solved by a commercial mixed-integer programming solver. If no better integer solution is found, the neighborhood is enlarged and the RP is solved again. The process is repeated until finding a better solution. In the remainder of the paper, we use the term *zooming* to refer to the search for an integer direction in the neighborhood of a fractional one.

To overcome the second limitation of ISUD, Zaghrouti et al. [22] proposed a second variant of ISUD, named I^{2}SUD, where an artificial variable is added to the SPP. Let $\mathbb{K}$ be an SPP, x^0 the current integer solution of $\mathbb{K}$ and $\mathcal{S} = \{j \in \mathbf{N} \,|\, x_j^0 \neq 0\}$. Let $\mathbb{K}'$ be the new SPP created from $\mathbb{K}$ by adding the artificial variable x_{n+1}, such that $x' = \begin{bmatrix} x & x_{n+1} \end{bmatrix}$, $c' = \begin{bmatrix} c & c_\mathcal{S} \end{bmatrix}$, $\mathbf{A}' = \begin{bmatrix} \mathbf{A} & e \end{bmatrix}$ and $\mathbf{N}' = \mathbf{N} \cup \{n+1\}$.

The cost $c_{\mathcal{S}}$ of variable x_{n+1} is equal to the cost of the current solution ($c_{\mathcal{S}} = \sum_{l \in \mathcal{S}} c_l$). The e column, whose components are equal to 1, is the result of aggregating the columns in $\mathcal{S}$. So, the current solution x^0 can also be represented by the following artificial solution: $x_{n+1} = 1$ and $x_j = 0$, $\forall j \in \mathbf{N}$. In this case, the relaxed CP model of $\mathbb{K}'$ is as follows:

$$(CP2) \qquad z^{CP2} = \min_{v,\lambda} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} c_j v_j + \sum_{l \in \mathcal{S}} c_l v_l - c_{\mathcal{S}} \lambda \tag{13}$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} a_{ij} v_j + \sum_{l \in \mathcal{S}} a_{il} v_l - \lambda = 0, \qquad \forall\, i \in \mathbf{T} \tag{14}$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l v_l = 1 \tag{15}$$

$$v \geq 0, \lambda \geq 0. \tag{16}$$

If $z^{CP2} < 0$, then a (possibly fractional) descent direction is found. As for CP1, the positive-valued variables v_j, $j \in \mathbf{N} \setminus \mathcal{S}$, identify the columns entering in the improved solution. On the other hand, the zero-valued variables v_l, $l \in \mathcal{S}$, correspond to the leaving variables. Because the directions found by CP2 are minimal with respect to the artificial solution, all extreme points in the convex hull of the feasible domain of $\mathbb{K}'$ are adjacent to this solution (see Proposition 3.2 in [22]). Consequently, CP2 can find an integer descent direction leading directly to an optimal solution of $\mathbb{K}$. The computational results reported in [22] show that I^{2}SUD outperforms ISUD and often finds an optimal solution by solving a single CP2.

The three algorithms ISUD, ZOOM and I^{2}SUD described above were tested on SPP instances where the columns are all generated a priori. This is generally not possible for very large-scale problem instances. For this reason, Tahir et al. [17] proposed a new primal algorithm, called integral column generation (ICG), for the SPP. It is based on a three-level decomposition: RP, CP and the CG PS. At each ICG iteration, the RMP is solved using ISUD to find a sequence of integer solutions. Then, a dual solution, said to correspond to the current integer solution, is used in the PS to generate new variables with a negative reduced cost. The authors showed that, if optimality is not yet reached, then at least one entering variable identified by an integer direction has a negative reduced cost with respect to any such dual solution. Computational results on CPP and VCSP instances show that ICG outperforms two CG heuristics commonly used in practice.

2.3 Main contributions

This paper is a continuation of the research work on the development of primal algorithms to efficiently solve the SPP as we introduce a new algorithm to efficiently solve the SPPSC. Below, we summarize its most important contributions.

- We introduce a new property of the SPP, called *pseudo-quasi-integrality*. Then, we show with an example that the side constraints in the SPPSC invalidate the quasi-integrality and pseudo-quasi-integrality properties of the SPP. Finally, we prove that it is possible to restore the pseudo-quasi-integrality property by increasing the size of the problem.
- We formulate new RP and CP which consider the side constraints.
- We present theoretical results that characterize the integer descent directions found by the new CP.
- Based on these theoretical results, we introduce the new primal algorithm I^{2}CG for solving the SPPSC.
- We test and compare two I^{2}CG versions on CPP instances involving up to 1740 flights.

3 Side constraints and quasi-integrality properties

In this section, we first show through an example that the SPPSC is not quasi-integral. Then, we introduce a new property (pseudo-quasi-integrality) of the SPP and discuss if it applies to the SPPSC.

Let us begin by defining the quasi-integrality property.

Definition 4 *A model* $\mathbb{Q}$ *is said to be* quasi-integral *if any edge of the convex hull of its integer solutions is an edge of the linear relaxation polyhedron of* $\mathbb{Q}$.

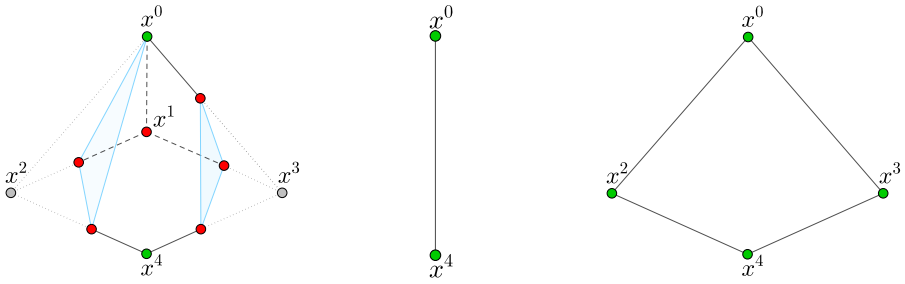
As mentioned earlier, quasi-integrality implies the existence of an integer path between any pair of integer extreme points of this polyhedron. Unfortunately, the side constraints in the SPPSC can invalidate this property as shown by the following example.

$$
\begin{array}{llccccccccccccccccccc}
(\mathbb{P}_1) & \min\limits_{x} & 12x_1 & + & 12x_2 & + & 12x_3 & + & 10x_4 & + & 10x_5 & + & 10x_6 & + & 2x_7 & + & 2x_8 & + & 2x_9 & & \\
& & x_1 & & & & & + & x_4 & & & & & + & x_7 & + & x_8 & & & = & 1 \\
& & x_1 & & & & & & & + & x_5 & & & + & x_7 & & & + & x_9 & = & 1 \\
& & & & x_2 & & & & & + & x_5 & & & + & x_7 & + & x_8 & & & = & 1 \\
& & & & x_2 & & & & & + & x_5 & & & & & + & x_8 & + & x_9 & = & 1 \\
& & & & & & x_3 & & & & & + & x_6 & & & & & & & = & 1 \\
& & & & & & x_3 & & & & & + & x_6 & & & & & & & = & 1 \\
& & & & & & x_3 & + & x_4 & + & x_5 & & & & & & & & & \leq & \frac{5}{2} \\
& & x_1 & + & x_2 & & & & & & & + & x_6 & & & & & & & \leq & 2 \\
& & & & & & & & & & & & & & & & & & x \in \{0,1\}^9 &
\end{array}
$$

For this example, we present some solutions of interest in Table 1, where x^0 is an initial integer solution and x^4 is the unique optimal solution. Observe that these two solutions are the only feasible ones and that we do not list all the extreme points of $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$. In Table 1, there are: the two feasible integer solutions x^0 and x^4, one fractional solution x^1 that is an extreme point of $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$, and two infeasible integer solutions x^2 and x^3 that would be feasible for the corresponding SPP, as shown in Figure 1. In this figure, the green and red nodes indicate integer and fractional extreme points of $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$, respectively. The grey nodes correspond to solutions x^2 and x^3 that are cut off by the side constraints in SPPSC. Note that two extreme points of $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$, corresponding to solutions x^i and x^j, are adjacent if and only if there exists a minimal direction d such that $x^i = x^j + d$ (see [6]).

Table 1: Some feasible and infeasible solutions of model $\mathbb{P}_1$

Solution	*Value*
$x^0 = (1,1,1,0,0,0,0,0,0)$	36
$x^1 = (0,0,1,0,0,0,\frac{1}{2},\frac{1}{2},\frac{1}{2})$	15
$x^2 = (1,1,0,0,0,1,0,0,0)$	34
$x^3 = (0,0,1,1,1,0,0,0,0)$	32
$x^4 = (0,0,0,1,1,1,0,0,0)$	30



i. $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$ **ii.** $conv(\mathcal{F}_{\mathcal{SPPSC}})$ **iii.** $conv(\mathcal{F}_{\mathcal{SPP}})$

Figure 1: $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$**,** $conv(\mathcal{F}_{\mathcal{SPPSC}})$ **and** $conv(\mathcal{F}_{\mathcal{SPP}})$ **of model** $\mathbb{P}_1$**.**

Figures 1.i and 1.ii show that the edge $\left(x^0, x^4\right)$ of $conv(\mathcal{F}_{\mathcal{SPPSC}})$ is not an edge of $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$. We can, thus, conclude that the SPPSC is not quasi-integral because of the side constraints. In addition, there is no integer path between x^0 and x^4 in $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$. This means that, from x^0, all the minimal descent directions that can be found by CP1 are fractional (or infeasible because it does not consider the side constraints). Therefore, ISUD can get stuck in the local minimum x^0 and never reach the optimal solution x^4.

This limitation can be overcome with the ZOOM algorithm, which can reach integer extreme points that are not necessarily adjacent to the current integer solution. However, before finding such a solution, ZOOM may need to increase the neighborhood several times, solving a relatively small binary linear program each time.

Quasi-integrality is a strong property that is not necessary in practice as primal algorithms only need to ensure the existence of an integer path between any pair of integer extreme points or, more precisely, between any initial integer solution and any optimal one. As we will show later, these paths can exist even if the problem is not quasi-integral. Let us present this requirement as the following property of a model.

Definition 5 *A model is said to be* pseudo-quasi-integral *when there is at least one integer path between any pair of integer extreme points of its linear relaxation polyhedron.*

As shown by the above example, the SPPSC does not possess the pseudo-quasi-integrality property. However, this property can be obtained by adding artificial edges to $\mathcal{F}_{\mathcal{SPPSC}^{\mathcal{LR}}}$. Indeed, inspired by I^{2}SUD, an artificial variable x_{n+1} can be added to model $\mathbb{P}$, yielding a new model denoted $\mathbb{P}'$. The cost assigned to variable x_{n+1} corresponds to the value of the current solution $\mathcal{S}$ (i.e., $c_{\mathcal{S}} = \sum_{l \in \mathcal{S}} c_l$). The constraint coefficients of variable x_{n+1} are equal to the right-hand sides of the constraints (2)–(3), i.e., $a_{i,n+1} = 1$, $\forall i \in \mathbf{T}$, and $q_{h,n+1} = b_h$, $\forall h \in \mathbf{H}$. By construction, $x'^0 = (0, 0, \ldots, 0, 1)$ is an integer solution of $\mathbb{P}'$. As shown in Section 4, this so-called *aggregated solution* is adjacent to all other integer extreme points of the linear relaxation polyhedron of model $\mathbb{P}'$. Therefore, it ensues the following proposition.

Proposition 1 *Model* $\mathbb{P}'$ *is pseudo-quasi-integral.*

Proof. Let x'^1 and x'^2 be any arbitrary pair of integer extreme points of the linear relaxation polyhedron of model $\mathbb{P}'$. Given that x'^0 is adjacent to both x'^1 and x'^2, x'^2 can be reached from x'^1 through the integer path $x'^1 - x'^0 - x'^2$. Therefore, model $\mathbb{P}'$ is pseudo-quasi-integral. □

Proposition 1 shows that, even if the side constraints in the SPPSC break the quasi-integrality and pseudo-quasi-integrality properties of the SPP, one can restore the pseudo-quasi-integrality by adding the artificial variable x_{n+1}. This variable also creates a shortcut, in the polyhedron, between the current solution and the optimal one.

4 Handling side constraints in ISUD

To solve the SPPSC, we develop an ICG algorithm, to be described in Section 5, where the RMP is solved by a new version of ISUD that can consider the side constraints. Here, we present how ISUD can be modified to handle the side constraints either in the RP or in the CP, or in both problems.

4.1 Handling side constraints in the RP

As mentioned earlier, the RP is formulated as a binary linear program. Therefore, the side constraints can easily be incorporated into its formulation to yield the following RP formulation:

$$(RP2) \quad \min_{x} \quad \sum_{j \in \mathbf{C}_{\mathcal{S}}} c_j x_j \tag{17}$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{C}_{\mathcal{S}}} a_{ij} x_j = 1, \qquad \forall\, i \in \mathbf{T}' \tag{18}$$

$$\sum_{j \in \mathbf{C}_{\mathcal{S}}} q_{hj} x_j \leq b_h, \qquad \forall\, h \in \mathbf{H} \tag{19}$$

$$x_j \in \{0,1\}, \qquad \forall\, j \in \mathbf{C}_{\mathcal{S}}. \tag{20}$$

Unlike RP1, the current integer solution S does not completely fill the basis when there are side constraints in RP2. Therefore, pivoting into the basis a first negative reduced cost variable does not ensure finding a better integer solution because of possible degeneracy. Nevertheless, compared to the original model $\mathbb{P}$, RP2 considers only a subset of its variables (the compatible ones) and a subset of its partitioning constraints (2), resulting in bases of reduced dimension. Consequently, the linear relaxations are easier to solve and the linear relaxation solutions contain less fractional-valued variables, resulting in a potentially much smaller branch-and-bound search tree to explore.

4.2 Handling side constraints in the CP

As discussed in Section 3, CP1 cannot handle the side constraints and, therefore, needs to be redefined. To do so, we consider model $\mathbb{P}'$ and the current integer solution $\mathcal{S}$ expressed in its aggregated form $x_{n+1} = 1$, $x_j = 0$, $\forall j \in \mathbf{N}$, which we also denote $\mathcal{S}' = \{n+1\}$. The corresponding CP, denoted CP3, is formulated as follows:

$$(CP3) \qquad z^{CP3} = \min_{v,\lambda} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} c_j v_j + \sum_{l \in \mathcal{S}} c_l v_l - c_{\mathcal{S}} \lambda \tag{21}$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} a_{ij} v_j + \sum_{l \in \mathcal{S}} a_{il} v_l - \lambda = 0, \qquad \forall\, i \in \mathbf{T} \tag{22}$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{hj} v_j + \sum_{l \in \mathcal{S}} q_{hl} v_l - \lambda b_h \leq 0, \qquad \forall\, h \in \mathbf{H} \tag{23}$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l v_l = 1 \tag{24}$$

$$v \geq 0, \lambda \geq 0, \tag{25}$$

where variable λ is associated with variable x_{n+1} and always takes a positive value in any feasible solution. A solution (v, λ) to CP3 defines a descent direction $(v, -\lambda)$ if $z^{CP3} < 0$. In this case, we can deduce from this direction the following descent direction $d = (d_j)_{j \in \mathbf{N}}$ for the original problem $\mathbb{P}$:

$$d_j = \begin{cases} \frac{v_j - \lambda}{\lambda} & \text{if } j \in \mathcal{S} \\ \frac{v_j}{\lambda} & \text{if } j \in \mathbf{N} \setminus \mathcal{S} \end{cases} \qquad \forall j \in \mathbf{N}. \tag{26}$$

One can easily verify that $c^\top d < 0$ and that $x^0 + d$ satisfies constraints (2) and (3) of $\mathbb{P}$. This direction d can, however, be integer or fractional according to Definition 1.

The following proposition characterizes the integrality of direction d in two different ways.

Proposition 2 *The following three statements are equivalent:*

1. *Descent direction d is integer;*
2. *$v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$;*
3. *For all pairs of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$, their columns A_{j_1} and A_{j_2} (excluding the side constraint coefficients) are disjoint.*

Proof. First, let us show that Statement 1 is true if and only if Statement 2 is true. ($\Rightarrow$) If d is integer, then, for $j \in \mathcal{S}$, $d_j = \frac{v_j - \lambda}{\lambda}$ must be in $\{0, -1\}$ because $x_j = 1$ and, for $j \in \mathbf{N} \setminus \mathcal{S}$, $d_j = \frac{v_j}{\lambda}$ must be in $\{0, 1\}$ because $x_j = 0$. In both cases, we deduce that $v_j \in \{0, \lambda\}$. ($\Leftarrow$) If $v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$, then it is easy to verify that $d_j \in \{0, -1\}$ if $j \in \mathcal{S}$ and $d_j \in \{0, 1\}$ if $j \in \mathbf{N} \setminus \mathcal{S}$. Therefore, $x^0 + d$ is integer and, thus, d is integer.

Second, let us show that Statement 2 is true if and only if Statement 3 is true. ($\Rightarrow$) If $v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$, then $A_{j_1}^\top A_{j_2} = 0$ for every pair of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$. Otherwise, for one of these pairs $j_1, j_2 \in \mathbf{N}$, there would exist a task $i \in \mathbf{T}$ such that $a_{ij_1} = a_{ij_2} = 1$ and the constraint (22) associated with task i would be violated ($\sum_{j \in \mathbf{N}} a_{ij} v_j \geq a_{ij_1} v_{j_1} + a_{ij_2} v_{j_2} = 2\lambda \neq \lambda$). ($\Leftarrow$) Consider an index $j^* \in \mathbf{N}$ such that $v_{j^*} > 0$ and a task $i \in \mathbf{T}$ such that $a_{ij^*} = 1$. If $A_{j_1}^\top A_{j_2} = 0$ for every pair of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$, then there is no other index $j' \neq j^*$ such that $v_{j'} > 0$ and $a_{ij'} = 1$. Consequently, from the constraint (22) associated with task i, we deduce that $\sum_{j \in \mathbf{N}} a_{ij} v_j = v_{j^*} = \lambda$. □

If direction d is integer, then the corresponding descent direction $(v, -\lambda)$ suggests to replace the current aggregated solution $\mathcal{S}'$ of problem $\mathbb{P}'$ by the solution $\mathcal{S}'' = \{j \in \mathbf{N} \mid v_j > 0\}$, i.e., the columns in $\mathcal{S}''$ must be pivoted in the solution while column of index $n+1$ must be pivoted out of it. The next proposition indicates that this direction is minimal.

Proposition 3 *Any descent direction $(v, -\lambda)$ found by CP3 which induces an integer direction d is minimal with respect to the aggregated solution $\mathcal{S}'$ of problem $\mathbb{P}'$.*

Proof. If d is integer, then $A_{j_1}^\top A_{j_2} = 0$ for every index pair $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$. Consequently, the column of index $n+1$ which has a coefficient of -1 for all set partitioning constraints cannot be pivoted out of the solution unless all columns in $\{j \in \mathbf{N} \mid v_j > 0\}$ are pivoted in the solution. □

The following theorem states that all improved solutions to $\mathbb{P}'$ are adjacent to the current aggregated solution x'^0 in the convex hull of the feasible domain of $\mathbb{P}'$, which is denoted $conv(\mathcal{F}_{\mathbb{P}'})$.

Theorem 1 *Let x'^* be a solution to $\mathbb{P}'$ whose cost is less than the cost of x'^0. Then, x'^* and x'^0 are adjacent extreme points in $conv(\mathcal{F}_{\mathbb{P}'})$.*

Proof. Let x^* and x^0 be the solutions corresponding to x'^* and x'^0 in problem $\mathbb{P}$. Denote by $d = x^* - x^0$ the integer descent direction that allows to move from x^0 to x^*. This direction d can be obtained from the following solution (v, λ) to CP3 (built using Proposition 2):

$$v_j = \begin{cases} \lambda & \text{if } d_j > 0 \text{ and } j \in \mathbf{N} \setminus \mathcal{S}, \\ \lambda & \text{if } d_j = 0 \text{ and } j \in \mathcal{S}, \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

$$\lambda = \Big(\sum_{j \in \mathbf{N} \setminus \mathcal{S} \mid d_j > 0} \alpha_j + \sum_{j \in \mathcal{S} \mid d_j = 0} \beta_j \Big)^{-1}. \tag{28}$$

According to Proposition 3, this solution yields a descent direction $(v, -\lambda)$ that is minimal. Therefore, x'^* is an extreme point of $conv(\mathcal{F}_{\mathbb{P}'})$ that is adjacent to x'^0. □

Note that this theorem can easily be extended to any solution x'^* of $\mathbb{P}'$ which does not have a cost less than that of x'^0.

In contrast to CP2, CP3 considers the side constraints of the original problem $\mathbb{P}$. Given that CP2 finds a direction to explore $\mathcal{F}_{\mathcal{SPP}\mathcal{LR}}$ and not $\mathcal{F}_{\mathcal{SPPSC}\mathcal{LR}}$, this direction might lead to an extreme point of $\mathcal{F}_{\mathcal{SPP}\mathcal{LR}}$ that is cut off by a side constraint of $\mathbb{P}$ (see Figure 1.i). The next proposition shows that this is not the case with CP3 when it returns an integer direction.

Proposition 4 *Let x be an integer extreme point of $\mathcal{F}_{\mathcal{SPP}\mathcal{LR}}$ that is cut off by a side constraint (3), say, the one indexed by $\hat{h} \in \mathbf{H}$. Then, no feasible solution (v, λ) to CP3 can yield the integer direction $d = x - x^0$ according to definition (26).*

Proof. Assume that there exists such a solution (v, λ). Because d is integer, we can deduce from (26) and Proposition 2 that $x_j = \frac{v_j}{\lambda}$, $\forall j \in \mathbf{N}$. If x violates side constraint (3) for index $\hat{h}$, then

$$\begin{aligned} & \sum_{j \in \mathbf{N}} q_{\hat{h}j} x_j > b_{\hat{h}} \\ \Leftrightarrow \quad & \sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{\hat{h}j} \frac{v_j}{\lambda} + \sum_{l \in \mathcal{S}} q_{\hat{h}l} \frac{v_l}{\lambda} > b_{\hat{h}} \\ \Leftrightarrow \quad & \sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{\hat{h}j} v_j + \sum_{l \in \mathcal{S}} q_{\hat{h}l} v_l > \lambda b_{\hat{h}}. \end{aligned}$$

This contradicts the feasibility of (v, λ) as it would violate the constraint (23) for index $\hat{h} \in \mathbf{H}$. Therefore, there exists no such solution (v, λ). □

Consequently, when the solution to CP3 induces an integer descent direction d, the ensuing solution $x^0 + d$ is always feasible for problem $\mathbb{P}$. Unfortunately, CP3 can also find solutions yielding a fractional direction d. In this case, we propose to search for an integer descent direction d, called a *subdirection of d*, as follows. Let $S_d^+ = \{j \in N \mid d_j > 0\}$ and $S_d^- = \{j \in N \mid d_j < 0\}$ be the indices of the columns that should enter and exit the current solution $\mathcal{S}$ according to direction d, respectively.

Definition 6 *A direction d^I is said to be a subdirection of direction d if*

$$d_j^I = 0, \quad \forall j \in N \setminus (S_d^+ \cup S_d^-), \qquad d_j^I \geq 0, \quad \forall j \in S_d^+, \qquad d_j^I \leq 0, \quad \forall j \in S_d^-.$$

Given that the sets S_d^+ and S_d^- are typically small, we solve a mixed-integer program (MIP) to find an integer descent subdirection d^I of direction d. This MIP aims at replacing some columns in the current solution by others, ensuring that the resulting solution is feasible. For each $h \in \mathbf{H}$, let $\hat{s}_h = b_h - \sum_{j \in \mathcal{S}} q_{hj}$ be the slack left in the side constraint h by the current solution $\mathcal{S}$. Furthermore, let y_j^+, $j \in S_d^+$, and y_j^-, $j \in S_d^-$, be binary variables associated with the columns in S_d^+ and S_d^-. Variable y_j^+ (resp., y_j^-) takes value 1 if the column indexed by j enters (resp., leaves) the current solution. The proposed MIP is:

$$z^{SD} = \min_{y^+, y^-} \quad \sum_{j \in S_d^+} c_j y_j^+ - \sum_{j \in S_d^-} c_j y_j^- \tag{29}$$

$$\text{s.t.:} \quad \sum_{j \in S_d^+} a_{ij} y_j^+ - \sum_{j \in S_d^-} a_{ij} y_j^- = 0, \qquad \forall\, i \in \mathbf{T} \tag{30}$$

$$\sum_{j \in S_d^+} q_{hj} y_j^+ - \sum_{j \in S_d^-} q_{hj} y_j^- \leq \hat{s}_h, \qquad \forall\, h \in \mathbf{H} \tag{31}$$

$$y_j^+ \in \{0,1\}, \quad \forall j \in S_d^+, \qquad y_j^- \in \{0,1\}, \qquad \forall j \in S_d^-. \tag{32}$$

Objective function (29) aims at minimizing the cost of replacing the selected columns in S_d^- by those selected in S_d^+. Constraints (30) impose that the entering columns cover exactly the same tasks as the

exiting columns. Finally, constraints (31) ensure that no side constraints are violated by this change. Note that the number of constraints can be reduced by considering only those for which there exists at least one column in $S_d^+ \cup S_d^-$ with a non-zero coefficient in this constraint.

An integer descent subdirection is found if $z^{SD} < 0$. This subdirection is given by

$$d_j^I = \begin{cases} y_j^+ & \text{if } j \in S_d^+ \\ -y_j^- & \text{if } j \in S_d^- \\ 0 & \text{otherwise.} \end{cases}$$

4.3 Possible RP-CP combinations

In this subsection, we discuss possible RP-CP combinations that can be used in ISUD to solve the SPPSC.

- **RP2-CP2**: This combination consists of handling the side constraints only in the RP. Since this problem involves subsets of constraints and columns, it can be easily solved using a commercial solver. However, the CP may find descent directions that do not point to a feasible region of $conv(\mathcal{F}_{\mathcal{SPPSC}})$, i.e., that points to a region where side constraints are violated. Therefore, to find an improved integer solution, it is often necessary to define a large neighborhood when zooming around these directions and the RP may, thus, consume more computational time.
- **RP1-CP3**: In this combination, the side constraints are only handled in the CP, ensuring that the descent directions found lead to a feasible region. However, if the directions found are fractional, the zooming procedure cannot be used since the RP does not consider the side constraints and, therefore, the solution process may terminate before reaching an optimal solution.
- **RP2-CP3**: With this combination, the advantages of handling the side constraints both in the RP and in the CP are exploited. In addition, the zooming procedure can be used in the RP to search for integer solutions in the neighborhood of any fractional direction found by the CP.

Given that the second combination offers no recourse when fractional directions are found, it is not considered in the following. We only compare the first and third combinations (i.e., RP2-CP2 and RP2-CP3) which rely on the zooming procedure to search for integer directions around fractional ones.

5 Improved integral column generation (I^{2}CG)

In this section, we describe the proposed I^{2}CG algorithm, an improved version of ICG [17], that can handle the side constraints of the SPPSC. Like ICG, it is based on a three-level decomposition. The first two levels correspond to the CG RMP which is decomposed into a RP and a CP. The third level consists of the CG PS which is, in our case, a shortest path problem with resource constraints that is solved by a labeling algorithm.

A pseudo-code for I^{2}CG is presented in Algorithm 1. It starts with an initial set of columns $\tilde{\mathbf{N}}$ (possibly artificial ones), an integer solution x^0 (also denoted $\mathcal{S}^0$) of cost z^0, and a corresponding dual solution π^0. Then, it tries to improve this initial solution by CG. At each CG iteration, the PS is solved in Step 3 and the RMP is solved by ISUD in Steps 7–16 and, if needed, by invoking the zooming procedure in Steps 17–28. Finally, Steps 29–32 count the number of successive CG iterations where the progress made in the objective value is deemed insufficient. This counter is used in the algorithm stopping criterion tested in Step 33. Note that, in this pseudo-code, we use RP2(σ) to specify that RP2 is built with respect to solution σ ($= \mathcal{S}$ or $\mathcal{S}'$), and CPversion($\mathbf{L}$) to identify the CP of version CPversion ($= CP2$ or $CP3$) defined by replacing set $\mathbf{N}$ with set $\mathbf{L}$. Below, we discuss the main parts of this pseudo-code in separate subsections.

Algorithm 1: I^{2}CG

input : initial columns $\tilde{\mathbf{N}}$, solution x^0 ($\mathcal{S}^0$), dual solution π^0, and cost z^0
output : best integer solution found x^*
parameter : CPversion, maxZoom, kZoom, minImprPerc, maxSuccFail, $\mathbf{K}^{CP} = \{k_1, \ldots, k_\eta\}$

$r \leftarrow 0$; $nbSuccFail \leftarrow 0$; $x^* \leftarrow x^0$; $\mathcal{S} \leftarrow \mathcal{S}^0$;
do
  Column generation
  $\mathbf{J} \leftarrow$ solve PS(π^r); $r \leftarrow r + 1$;
  if $\mathbf{J} \neq \emptyset$ **then**
    $\tilde{\mathbf{N}} \leftarrow \tilde{\mathbf{N}} \cup \mathbf{J}$;
    RP-CP loop
    $k \leftarrow k_1$;
    do
      $z^r \leftarrow$ solve RP2($\mathcal{S}$);
      do
        $(d, \pi^r) \leftarrow$ solve CPversion($\tilde{\mathbf{N}}^k_{\mathcal{S}}$);
        if *d is integer **or** contains an integer subdirection* **then**
          update x^* and $\mathcal{S}$;
          break;
        $k \leftarrow nextPhase(k, \mathbf{K}^{CP})$;
      while $k \neq 0$;
    while *d is integer **or** contains an integer subdirection*;
    Zooming
    if *d is fractional* **then**
      $\mathcal{S}' \leftarrow \mathcal{S}$; $d^1 \leftarrow d$;
      for $\ell = 1$ ***to*** $maxZoom$ **do**
        $\mathcal{S}' \leftarrow \mathcal{S}' \cup supp(d^\ell)$;
        $z^r \leftarrow$ solve RP2($\mathcal{S}'$);
        if *improved integer solution found* **then**
          update x^* and $\mathcal{S}$;
          break;
        if $\ell < maxZoom$ **then**
          $d^{\ell+1} \leftarrow$ solve CPversion($\tilde{\mathbf{N}}^{kZoom}_{\mathcal{S}'}$);
          if $d^{\ell+1} = d^\ell$ **then**
            break;
    Control
    if $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImprPerc$ **then**
      $nbSuccFail \leftarrow nbSuccFail + 1$;
    else
      $nbSuccFail \leftarrow 0$;
while $J \neq \emptyset$ ***and*** $nbSuccFail < maxSuccFail$;

5.1 Solving the pricing subproblem

In Step 3, the PS (which may be separated in multiple PSs) is solved by, e.g., a labeling algorithm to find a set $\mathbf{J}$ of negative reduced cost columns, where the reduced cost of column j is computed as $\bar{c}_j = c_j - \sum_{t \in \mathbf{T}} \pi_t a_{tj} - \sum_{h \in \mathbf{H}} \pi_h q_{hj}$. If $\mathbf{J}$ is empty, the algorithm stops. Otherwise, the columns in $\mathbf{J}$ are added to the current RMP column pool $\tilde{\mathbf{N}}$ in Step 5. Note that a multi-phase strategy that initially restricts the PS solution space and gradually enlarges it when no "good" negative reduced columns can be found is also applied when solving the PS. Details about this strategy are provided at the end of Section 5.2.

For both versions of the CP, the dual solution $\pi^r = \left((\pi^r_t)_{t \in \mathbf{T}}, (\pi^r_h)_{h \in \mathbf{H}}\right)$ used in the PS at iteration r is either provided by the initial dual solution π^0 (when $r = 0$) or by optimal dual values obtained in the

last solution of RP2 in Step 8 or the CP in Step 10. More precisely, if $CP2$ is used in I^{2}CG, then the duals π_t^r, $\forall t \in \mathbf{T}$, and π_h^r, $\forall h \in \mathbf{H}$, are set equal to those of constraints (14) and (19), respectively. This choice of dual solution is based on empirical results and offers no guarantee of algorithmic convergence. On the other hand, when I^{2}CG relies on $CP3$, both sets of dual values are provided by $CP3$, namely, from constraints (22) and (23). In this case, the following proposition can be proven.

Proposition 5 *Let $\mathcal{S}$ be an integer solution with cost $c_{\mathcal{S}} = \sum_{j\in\mathcal{S}} c_j$. Let $\mathcal{S}^*$ be an improved solution ($\sum_{j\in\mathcal{S}^*} c_j < c_{\mathcal{S}}$). At least one variable x_j, $j \in \mathcal{S}^*$, has a negative reduced cost with respect to the dual solution $\pi \in \mathbb{R}^{|\mathbf{T}|+|\mathbf{H}|}$ provided by (22) and (23).*

Proof. After solving CP3, the reduced cost of λ is always equal to 0, which implies

$$c_{\mathcal{S}} = \sum_{i\in\mathbf{T}} \pi_i + \sum_{h\in\mathbf{H}} b_h \pi_h. \tag{33}$$

Furthermore, given that $\pi_h \leq 0$ and $\sum_{j\in\mathcal{S}^*} q_{hj} \leq b_h$ for all $h \in \mathbf{H}$, the following relation holds:

$$\sum_{h\in\mathbf{H}} \sum_{j\in\mathcal{S}^*} q_{hj}\pi_h \geq \sum_{h\in\mathbf{H}} b_h \pi_h. \tag{34}$$

Now, let us assume that all variables x_j, $j \in \mathcal{S}^*$, have a non-negative reduced cost, i.e.,

$$\bar{c}_j = c_j - \sum_{t\in\mathbf{T}} \pi_t a_{tj} - \sum_{h\in\mathbf{H}} \pi_h q_{hj} \geq 0, \qquad \forall j \in \mathcal{S}^*. \tag{35}$$

These inequalities yield

$$\begin{aligned}
\sum_{j\in\mathcal{S}^*} (c_j - \sum_{i\in\mathbf{T}} a_{ij}\pi_i - \sum_{h\in\mathbf{H}} q_{hj}\pi_h) \geq 0 \quad &\Rightarrow \quad \sum_{j\in\mathcal{S}^*} c_j \geq \sum_{i\in\mathbf{T}} \pi_i + \sum_{j\in\mathcal{S}^*} \sum_{h\in\mathbf{H}} q_{hj}\pi_h \\
&\Rightarrow \quad \sum_{j\in\mathcal{S}^*} c_j \geq \sum_{i\in\mathbf{T}} \pi_i + \sum_{h\in\mathbf{H}} b_h \pi_h \qquad \text{(from (34))} \\
&\Rightarrow \quad \sum_{j\in\mathcal{S}^*} c_j \geq c_{\mathcal{S}} \qquad\quad \text{(from (33))}.
\end{aligned}$$

This contradicts the hypothesis that $\mathcal{S}^*$ is an improved solution. Therefore, the above assumption is false and at least one variable x_j, $j \in \mathcal{S}^*$, has a negative reduced cost. □

The following corollary, which provides an optimality certificate, directly ensues from this proposition.

Corollary 1 *If $z^{CP3} = 0$ and no negative reduced cost columns are generated when solving the PS using the dual solution $\pi \in \mathbb{R}^{|\mathbf{T}|+|\mathbf{H}|}$ provided by (22) and (23), then the current solution $\mathcal{S}$ is optimal for $\mathbb{P}$.*

If $z^{CP} < 0$ and $\mathcal{S}$ is not optimal (i.e., there exists an improved solution $\mathcal{S}^*$), then there is no guarantee that the PS can generate a negative reduced cost column. Indeed, even if Proposition 5 stipulates that there exists at least one variable x_j, $j \in \mathcal{S}^*$, that has a negative reduced cost, all negative reduced cost variables may belong to the set $\tilde{\mathbf{N}}$ of already generated columns. In this case, nonnegative reduced cost columns have to be generated by the PS and added to $\tilde{\mathbf{N}}$ to find the corresponding descent direction. Identifying these columns does not seem feasible. Alternatively, to ensure the exactness of the algorithm, one can restrict CP3 by including the disjonctive constraints (12) and generating negative reduced cost columns when branching to satisfy them. Given that we do not necessarily look for optimal solutions in our computational experiments, this option has not been implemented. Furthermore, our computational results show that this situation is not frequent, at least not before being very close to optimality.

5.2 RP-CP loop

In the RP-CP loop (Steps 7–16), ISUD is applied to solve the RMP with the goal of finding the best integer solution given the available set of columns $\tilde{\mathbf{N}}$. This loop stops, however, when no integer direction or subdirection is found by the CP. Note that the CP version used is passed to the algorithm as a parameter ($CPversion$). In the remainder of this paper, we denote by I^2CG_2 and I^2CG_3 the I^2CG algorithm based on CPversion $= CP2$ and CPversion $= CP3$, respectively.

Theoretically, in a given CG iteration r, an optimal solution to the RMP can be found by solving a single CP3 because this solution is adjacent to the aggregated solution (see Theorem 1). However, in practice, several iterations in the RP-CP loop may be necessary to reach this solution because of the objective function of CP3, which minimizes an "average" reduced cost. Indeed, one can observe that the cost of a solution (v, λ) to CP3 yielding an integer descent direction d is equal to $\lambda c^\top d$, where d and λ are defined by (27)–(28). In particular, if the weights α_j and β_l in (24) are all equal to 1, then $\lambda = 1/n^+(v)$, where $n^+(v)$ is equal to the number of v_j variables with a positive value, i.e., the number of columns in the solution $x^0 + d$. In this case, the cost of (v, λ) is equal to the average reduced cost per column in this solution.

For the SPP, Zaghrouti et al. [22] have shown that there exists weights α_j^* and β_l^* such that the direction induced by the corresponding solution of CP2 leads to an optimal solution of the problem. Such result can be generalized to CP3 and the SPPSC RMP. Unfortunately, there is no method to identify such weights a priori.

For our tests, we used the following weights: $\alpha_j = \delta^j$, $j \in \mathbf{N} \setminus \mathbf{C}_\mathcal{S}$, $\alpha_j = 2$, $j \in \mathbf{C}_\mathcal{S} \setminus \mathcal{S}$, and $\beta_l = 1$, $l \in \mathcal{S}$, where δ^j is the degree of incompatibility of column A_j. This incompatibility degree can be mathematically defined as the "distance" between column A_j and the vector subspace generated by the columns in $\mathcal{S}$ (omitting the side constraint coefficients); a formal definition is given below. Using these weights, CP3 aims at minimizing the average reduced cost per degree of incompatibility in the columns of the resulting solution $x^0 + d$. In fact, empirical results show that improved integer solutions are often composed of columns with a small degree of incompatibility. Thus, this objective function favors finding them.

The degree of incompatibility of a column A_j, $j \in \mathbf{N}$, is defined according to the current solution $\mathcal{S}$ and a so-called compatibility matrix M.

Definition 7 *A matrix M is a compatibility matrix if and only if $||MA_j||_1 = 0$ for every compatible column A_j, $j \in C_\mathcal{S}$.*

Definition 8 *The incompatibility degree of a column A_j, $j \in \mathbf{N}$, is $\delta^j = ||MA_j||_1$.*

Note that $\delta^j = 0$ for all compatible columns A_j, $j \in C_\mathcal{S}$.

In our algorithm, we use the compatibility matrix M_2 of Bouarab et al. [4] which is designed for vehicle routing and crew scheduling applications. With this matrix, the degree of incompatibility of a column A_j indicates the number of times that the task subsets of the current solution $\mathcal{S}$ must be divided to ensure that A_j becomes compatible. For example, if two columns in $\mathcal{S}$ cover the task subsets $\{1, 2, 3, 4\}$ and $\{5, 6\}$, then $\delta^j = 2$ for an incompatible column A_j covering the task subset $\{4, 5\}$ because, to make A_j compatible, both subsets $\{1, 2, 3, 4\}$ and $\{5, 6\}$ must be divided to yield the subsets $\{1, 2, 3\}$, $\{4\}$, $\{5\}$, and $\{6\}$.

To accelerate Algorithm 1 and further favor the identification of improved integer solutions made up of columns with a small degree of incompatibility, a multi-phase strategy similar to those developed by El Hallaoui et al. [8] for the DCA algorithm and adapted by Zaghrouti et al. [21] for the ISUD algorithm is implemented in Steps 9–15. Let $\mathbf{K}^{CP} = \{k_1, k_2, \ldots, k_\eta\}$ be an ordered set of phases, where k_i, $i = 1, 2, \ldots, \eta$, is a maximum degree of incompatibility, $k_i \geq 1$ and $k_i < k_{i+1}$ for all $i = 1, 2, \ldots, \eta-1$. For example, $\mathbf{K}^{CP}$ may be equal to $\{1, 2, 3, 5, 8\}$. In a phase $k \in \mathbf{K}^{CP}$, the CP solved in Step 10 is restricted to a subset $\tilde{\mathbf{N}}_\mathcal{S}^k$ of the generated columns, namely, those with an incompatibility degree less

than k with respect to solution $\mathcal{S}$. Therefore, every subset $\tilde{\mathbf{N}}^k$, $k \in \mathbf{K}^{CP}$, contains all compatible columns and subset $\tilde{\mathbf{N}}^{k_\eta} = \tilde{\mathbf{N}}$ if k_η is sufficiently large.

Each CG iteration starts in phase $k = k_1$. Then, after solving the RP in Step 8, the search for a descent direction starts in the current phase k. If the computed direction d in Step 10 is integer or contains an integer subdirection, then the best found solution is updated. Otherwise, in Step 14, function *nextPhase* identifies the next phase to explore in set $\mathbf{K}^{CP}$ if $k < k_\eta$ and returns $k = 0$ otherwise. In the former case, the CP is solved again. In the latter, the algorithm continues with the zooming procedure.

As mentioned in the previous section, we also implemented a multi-phase strategy for the PS that is independent from the one described above, using a set of phases $\mathbf{K}^{PS}$. The objective of this strategy is to generate, at the beginning of the algorithm, columns that are slightly incompatible with the current best integer solution. In phase $k \in \mathbf{K}^{PS}$, the PS (a shortest path problem with resource constraints) involves an additional resource constraint that limits the incompatibility degree of the generated columns to be less than or equal to k (see El Hallaoui et al. [7] for further details). The algorithm starts in the first phase in $\mathbf{K}^{PS}$. It moves to the next phase either when no column is generated in Step 3 (in which case, the PS is immediately solved again) or when a failure is identified in Step 29.

5.3 Zooming

When the last direction d found by the CP in Step 10 is fractional, the *zooming* procedure (Steps 18–28) is called to look for an improved solution in the neighborhood of this direction. The first neighborhood is created by adding the column indices j such that $d_j > 0$ to the index set $\mathcal{S}$ to form an augmented index set $\mathcal{S}'$ (Step 20). Then, RP2 defined with the variables compatible with $\mathcal{S}'$ is solved in Step 21 with the hope of finding an improved integer solution. If this is the case, the best solution found is updated and the zooming procedure stops. Otherwise, the neighborhood is enlarged by solving the CP (Step 26) to find a new descent direction and increase set $\mathcal{S}'$, before solving again the RP. In total, at most $maxZoom$ neighborhoods are explored before halting the zooming procedure. To keep the size of the CP relatively small and favor finding improved solutions composed of columns with a small degree of incompatibility, the CP is defined for the set $\tilde{\mathbf{N}}_{\mathcal{S}'}^{kZoom}$ of columns, i.e., those having a degree of incompatibility with respect to $\mathcal{S}'$ less than or equal to the value of parameter $kZoom$. Note that, because set $\mathcal{S}'$ increases at each iteration, the size of $\tilde{\mathbf{N}}_{\mathcal{S}'}^{kZoom}$ increases at each iteration and the degree of incompatibility of its columns with respect to the current solution $\mathcal{S}$ can be larger than $kZoom$. Nevertheless, the direction $d^{\ell+1}$ found at iteration ℓ may be the same as d^ℓ, the one obtained in the previous iteration. In this case (Step 27), set $\mathcal{S}'$ cannot increase and the zooming procedure stops.

5.4 Control and stopping condition

Once the current CG iteration is completed, i.e., after solving the PS and the RMP through the RD-CP loop and, possibly, the zooming procedure, the improvement in the cost of the best solution found in this iteration is evaluated in Step 29 of Algorithm 1. If this improvement in percentage does not exceed the value of parameter $minImprPerc$, then this CG iteration is deemed a failure and the number of successive iterations with a failure is increased by 1 (Step 30). Otherwise, it is qualified as a success and the number of successive iterations with a failure is reset to 0 in Step 32.

The main while loop stops in Step 33 when the PS cannot generate negative reduced cost columns or when a failure occurs in $maxSuccFail$ successive iterations.

6 Computational results

In this section, we present the results of the computational tests that we performed on real-life instances of the CPP with up to 1740 flights. Each instance (derived from the datasets of Kasirzadeh et al. [11])

concerns a single aircraft type with a horizon of one week and three bases. To each instance, we added 21 (7 days times 3 bases) side constraints to limit the number of pilots available per base and day. For each constraint $h \in \mathbf{H}$ and pairing $j \in \mathbf{N}$, coefficient q_{hj} is equal to 1 if j and h are associated with the same base and j covers a flight during the day associated with h. Furthermore, the right-hand side b_h of constraint $h \in \mathbf{H}$ corresponds to the number of pilots available in the base associated with h. For our tests, set $\mathbf{N}$ is initialized with artificial columns, namely, one for each task $i \in \mathbf{T}$ with no contribution to the side constraints and a large cost M. These columns form the initial solution $\mathcal{S}^0$ which has a cost $z^0 = M|\mathbf{T}|$. The dual solution π^0 is thus given by $\pi_i^0 = M$, $\forall i \in \mathbf{T}$, and $\pi^0 = 0$, $\forall h \in \mathbf{H}$.

In our experiments, we tested both versions of I^{2}CG (I^{2}CG$_2$ and I^{2}CG$_3$), a diving heuristic (DH), and a restricted master heuristic (RMH) (see Joncour et al. [10]). DH is a BP heuristic which explores a single branch of the search tree in a depth-first fashion without any backtracking. After solving each linear relaxation by CG, it fixes the pairing variable with the highest fractional value to 1 until finding an integer solution. RMH starts by solving the linear relaxation of the SPPSC by CG and converts the last RMP into a MIP which is then solved by a commercial solver. It should be noted that DH and RMH do not guarantee to find an integer solution. In fact, as no backtracking is allowed in DH, the algorithm may end up with an infeasible RMP after fixing a certain number of variables to 1. For RMH, the subset of columns generated for solving the linear relaxation of the problem may simply not contain a feasible integer solution. For both I^{2}CG$_2$ and I^{2}CG$_3$, the following parameter values were used: $minImprPerc = 0.0025$, $maxSuccFail = 9$, $kZoom = 5$, $\mathbf{K}^{CP} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathbf{K}^{PS} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Furthermore, we set $maxZoom = 4$ for I^{2}CG$_2$ and $maxZoom = 2$ for I^{2}CG$_3$. All these values were determined based on preliminary test results.

All our tests were run on a Linux machine equipped with an Intel i7 processor clocked at 3.4 GHz. In all algorithms, the SPs were solved using dynamic programming implemented using the Boost library (version 1.54). All linear RMPs and MIPs were solved by CPLEX (version 12.6).

This section is divided in three parts. First, we compare the performance of the algorithms I^{2}CG$_2$, I^{2}CG$_3$, DH, and RMH. Second, we present detailed results to provide insights on the performance of I^{2}CG$_3$, compared to DH and RMH. Finally, we study the impact of the side constraint right-hand side values on the computational times of all algorithms.

6.1 Performance tests

In Table 2, we report computational results that allow to compare the performance of the algorithms I^{2}CG$_2$, I^{2}CG$_3$, DH, and RMH. Each instance is identified by an Id (Id) and the number of tasks it involves ($Tasks$). Then, for each algorithm, we read from left to right: the total computational time in seconds ($Time$), the optimality gap in percentage between the cost of the best solution found and the linear relaxation optimal value (Gap), the number of column generation iterations ($Itr.$), the total number of integer solutions found ($IntS$), the percentage of saturated side constraints in the best solution found ($Sc.$), and the total number of generated columns ($Cols$). For DH, the number of integer solutions is not indicated, as it is always equal to 1. It should be noted that the linear relaxation optimal values were only computed to report the optimality gaps, so the time required to calculate them is not included in the total time of the I^{2}CG algorithms. In the last row of the table, we report the average of each table column. For each instance, the least computational time and the least gap among all algorithms are highlighted in bold.

From these results, we observe that I^{2}CG$_2$ finds integer solutions with a slightly higher average optimality gap than those obtained with I^{2}CG$_3$. This is not surprising because I^{2}CG$_2$ does not handle the side constraints in CP2 and the descent directions found by CP2 can lead to infeasible regions. So, to increase the probability of finding integer solutions in the neighborhood of these directions via the zooming procedure, I^{2}CG$_2$ must search in larger neighborhoods than I^{2}CG$_3$. This justifies: i) setting $maxZoom = 4$ for I^{2}CG$_2$ instead of $maxZoom = 2$ for I^{2}CG$_3$; ii) larger computational times for I^{2}CG$_2$ except for one instance.

Table 2: Results of I^2CG_2, I^2CG_3, DH and RMH.

Instance		I^2CG_2						I^2CG_3						*DH*				*RMH*					
Id	*Tasks*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*
727	239	9	0.36	23	73	33	23413	8	**0.35**	20	66	42	19848	9	**0.35**	59	33	9474	**5**	0.42	21	5	33	8674
09	348	13	0.18	14	56	9	24408	13	**0.16**	19	56	9	28438	5	0.36	26	9	10695	**4**	0.20	17	5	9	10645
94	424	44	0.21	19	93	66	50823	36	**0.19**	18	101	61	57028	38	0.45	71	66	19615	**15**	0.26	23	5	71	18184
95	1255	2411	**0.17**	24	455	42	457784	**1669**	**0.17**	23	489	42	419540	8052	0.22	903	52	171415	7640	1.19	53	30	52	103250
757	1290	915	**0.01**	17	324	4	326725	**855**	**0.01**	17	402	4	337999	3479	0.03	756	14	105320	1163	0.16	75	8	19	62791
319	1293	**1482**	0.17	20	488	42	277853	1542	**0.11**	23	462	47	283315	4514	0.24	1170	42	133403	7406	2.14	89	19	76	77687
320	1740	2034	0.06	24	438	38	456921	1874	**0.05**	17	565	33	369921	8126	0.08	1317	38	151183	**1144**	0.15	103	13	42	91577
	Avg.	987	0.17	20	275	33	231132	**857**	**0.15**	20	306	34	216584	3460	0.24	614	36	85872	2482	0.64	54	12	43	53258

We can also observe that I^2CG_3 generally finds better solutions faster than other methods. Indeed, the solution found by I^2CG_3 has an optimality gap that can be 3 times smaller than the gap of the solution found by DH. In addition, I^2CG_3 reduces the DH time by a factor of 3 for instance 319 and of 4 for instances 95, 757 and 320. On average, I^2CG_3 is 4 times faster than DH, 2.8 times faster than RMH, and improves the optimality gap by a factor of 1.6 for DH and of 4 for RMH. We also notice that, on average, I^2CG_3 generates 2.5 times more columns than DH. This can be explained by the extra resource that computes the incompatibility degree in the I^2CG_3 SPs (see Section 5.2) which reduces label dominance and also by the absence of dominance at the network sink nodes. This choice of generating a large number of columns was justified in Tahir et al. [17], where the authors showed the interest of this strategy for ICG that is, at the opposite, not beneficial for DH. In fact, the large number of generated columns can be efficiently managed by I^2CG in the RMP using the multi-phase strategy (see Section 5.2). This speed up can also be explained by the number of iterations required by I^2CG_3 (20 on average), which is negligible compared to that of DH (614 on average).

Another remarkable characteristic of the I^2CG_3 algorithm is the large number of integer solutions found throughout the solution process. Indeed, I^2CG_3 finds 33 integer solutions per iteration for the largest instance 320 and 15 integer solutions per iteration on average over all instances. This corresponds to an improved solution at every 3 seconds on average. This characteristic of I^2CG_3 is highly desirable in the industry where the solution can sometimes be stopped prematurely because of a deadline for producing the crew schedules.

Figure 2 illustrates the evolution of the cost of the current solution as a function of the computational time for instance 320 (a similar behavior is observed for the other instances). It includes an enlargement of the function between times 1000 and 8100 seconds. Each blue spot indicates an integer solution found by I^2CG_3. The red diamond and the green star represent the unique solution found by DH and the best one computed by RMH. This figure shows the fast improvement of the current solution at the beginning of the I^2CG_3 solution process. Like other column generation methods, a tailing off occurs, i.e., the current cost slowly improves in the second half of this process. We also note that, for I^2CG_3, integer solutions are found regularly, which means that I^2CG_3 is not prone to degeneracy.

In Tables 3 and 4, we detail the I^2CG_3, DH and RMH results reported in Table 2. The objective is to analyse the behavior of different components of the three algorithms. In these tables, we give for each algorithm the time consumed by each algorithm component in seconds ($Time$), the number of times that the PSs were solved ($Itr.$), i.e., the number of CG iterations, and the average number of generated columns per iteration ($Av.GCol$). For I^2CG_3, we also report the number of integer solutions found by RP2 ($IntS$), the average number of constraints in RP2 ($Av.Cst.$), the average number of non-zero coefficients ($Av.Nz.$) in the RP2 constraint matrix, the number of solved RP2s ($No.$), the number of integer directions found by CP3 ($IntD$), the number of integer subdirections ($IntSD$), and the average number of columns in CP3 ($Av.Cols.$). For RMH, we also provide the number of non-zero coefficients in the MIP ($Nz.$). Averages of these results are given in the last row of each table. Recall that I^2CG_3 can find an integer solution by solving either RP2 or CP3.

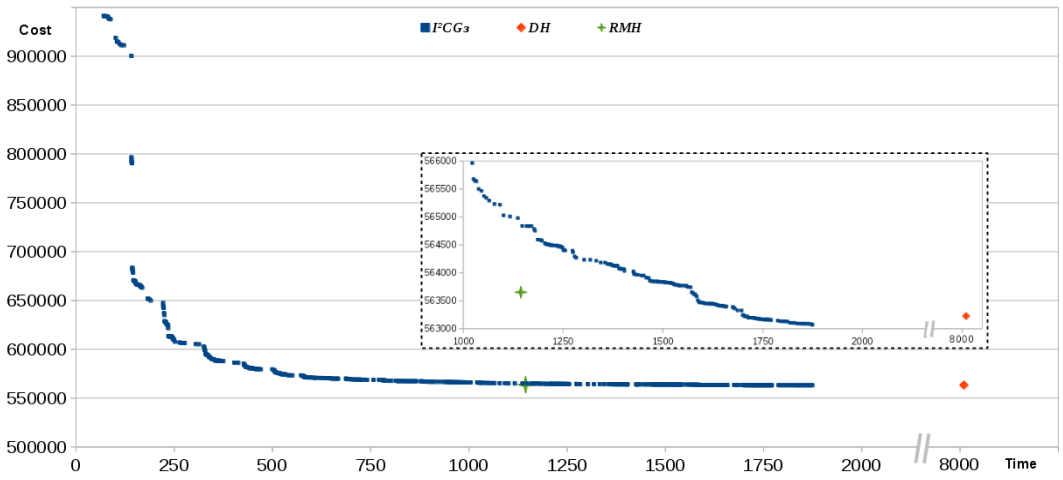


Figure 2: Cost of the current solution in function of the computational time – Instance 320.

6.2 Some insights

Table 3: Detailed results of I^2CG_3.

Instance		I^2CG_3												
		RP2					*CP3*					*PS*		
Id	*Tasks*	*Time*	*IntS*	*Av.Cst.*	*Av.Nz.*	*No.*	*Time*	*intD*	*intSD*	*Av.Col.*	*No.*	*Time*	*Itr.*	*Av.GCol.*
727	239	1	25	115	2997	90	3	24	17	1389	223	2	20	992
09	348	1	9	179	3967	88	7	17	30	2696	244	3	19	1497
94	424	6	28	206	6146	111	18	33	42	5190	235	7	18	3168
95	1255	335	112	546	18096	302	983	43	334	29344	456	270	23	18241
757	1290	8	46	689	10472	206	691	3	353	33096	310	129	17	19882
319	1293	423	130	539	13497	258	941	9	323	25998	399	127	23	12318
320	1740	292	203	806	15222	256	1337	21	341	37580	359	178	17	21760
	Avg.	152	79	440	10056	187	568	21	205	19327	318	102	19	11123

Table 4: Detailed results of DH and RMH.

Instance		*DH*				*RMH*					
		RMP	*PS*			*RMP*	*MIP*		*PS*		
Id	*Tasks*	*Time*	*Time*	*Itr.*	*Av.GCol.*	*Time*	*Time*	*Nz.*	*Time*	*Itr.*	*Av.GCol.*
727	239	2	7	59	160	1	1	81336	2	21	413
09	348	1	4	26	411	1	1	102728	2	17	626
94	424	5	32	71	276	2	3	180240	8	23	790
95	1255	430	7605	903	189	60	7003	1112762	576	53	1948
757	1290	299	3169	756	139	74	771	533957	318	75	837
319	1293	468	4024	1170	114	97	7001	748058	307	89	872
320	1740	706	7401	1317	114	194	366	842719	583	103	889
	Avg.	273	3177	614	200	61	2163	514542	256	54	910

From Table 3, we observe that CP3 consumes on average more than 70% of the total computational time of I^2CG_3, with an average of 2 seconds per CP3 solved. It finds more than 70% of the integer solutions identified by the algorithm. This is not surprising as the descent directions found by CP3 can reach any improved integer solution of the SPPSC according to Theorem 1. Integer solutions obtained from CP3 are derived from an integer descent direction in 10% of the times and from an integer descent subdirection in 90% of the times. We also observe that RP2 requires less than 1 second per resolution. This can be explained by its average number of constraints that does not exceed one-half of the number

of constraints in the original problem. Finally, note that CP3 contains an average number of columns that is not very large, which makes it relatively easy to solve at each iteration.

From Table 4, we make the following observations. In DH and RMH, the average time consumed by the PS per CG iteration is less than that used in I^2CG_3 for the largest instances. This is mainly due to the incompatibility resource added in the PS of I^2CG_3. Nevertheless, the PS in DH takes, on average, more than 91% of the total time, while it requires only 13% of it in I^2GC_3. This difference is mainly explained by the number of calls to the PS and the fact that the RMP is a linear program in DH while it is an integer program solved by ISUD in I^2GC_3. For RMH, we see that the MIP consumes on average more than 80% of the total time. The average size of the solved MIPs is very large compared to the average size of the RPs solved in I^2GC_3 (see Table 2). Finally, observe that the average number of columns generated per iteration in DH is much less than in RMH. This is explained by the large number of columns generated in the first CG iterations and that RMH stops generating columns after solving the linear relaxation.

To conclude this section, we briefly discuss the convergence of the minimum reduced cost in the I^2CG_3 algorithm. For instance 320, Figure 3 depicts the minimum reduced cost computed at each CG iteration (by solving the PSs) for I^2CG_3 (left) and DH (right). These results show that the dual variables used to generate columns in I^2CG_3 are more stable than those used by DH. In fact, every time that variables are fixed in DH, some of the dual variables are highly perturbed, yielding the generation of columns that may have little chance to be part of the final integer solution. Such behavior is not observed for I^2CG_3, which may explain its rapid convergence.

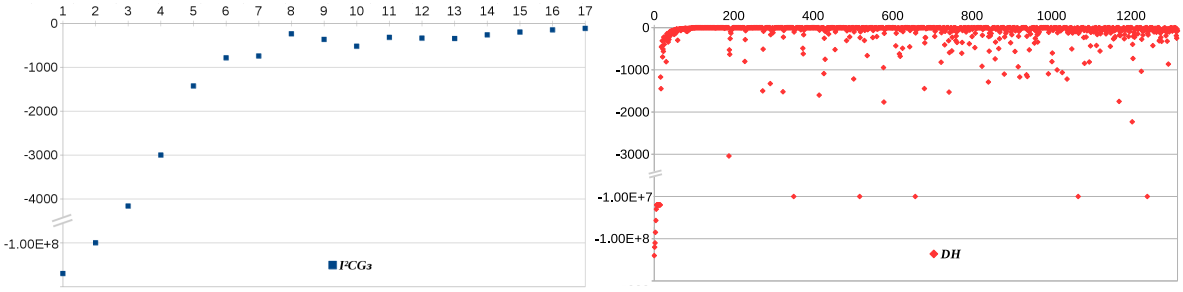


Figure 3: Minimum reduced cost at each CG iteration for I^2CG_3 and DH – Instance 320.

6.3 Sensitivity of I^2CG_3 to right-hand side values

To analyze the sensitivity of the I^2CG_3 algorithm to the tightness level of the side constraints and validate that it still outperforms the other algorithms for various levels, we conducted another series of experiments with additional instances derived from the instances used for our main tests. For each of them, we created four other instances: one with larger supplementary constraint right-hand side values (i.e., with less tight constraints) and three with increasingly smaller values. The tightness levels are numbered from 1 (loosest) to 5 (tightest), with level 2 corresponding to the original level. The average results of these experiments are reported in Table 5, where column *Level* indicates the tightness level of the instance and the other columns have the same meaning as in Table 2. Note that, for level 2, we report the same average results as in Table 2. Instance-by-instance results can be found in Appendix A.

Table 5: Sensitivity analysis average results.

	I^2CG_2						I^2CG_3						DH					RMH					
Level	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*
1	969	0.14	20	274	15	215534	737	0.12	19	294	17	215109	3605	0.23	651	15	83596	2374	0.39	54	13	21	50950
2	987	0.17	20	275	33	231132	857	0.15	20	306	34	216584	3460	0.25	615	36	85872	2482	0.65	54	12	43	53258
3	1112	0.35	24	275	55	243027	933	0.23	21	308	55	228903	3824	0.58	713	59	92079	3490	1.21	55	26	61	54303
4	986	0.49	27	290	64	291053	1011	0.32	22	308	67	233928	4087	0.51	827	74	99218	4339	0.88	56	12	68	55486
5	1052	0.82	27	303	77	292632	1233	0.41	24	310	77	258582	4388	0.77	893	78	104021	4389	1.02	59	13	79	57122

These results show that I^2CG_3 always finds the best solution in, generally, less computational time than the other algorithms for all tightness levels. The tightness of the side constraints has an impact on the optimality gap for most instances. Indeed, when many of those constraints are active (see their average percentage in column *Sc.*), they cut several interesting regions close to the best integer solutions and create many fractional extreme points. However, we can easily see that the effect of tight side constraints differs from one method to another. RMH is the mostly affected algorithm because it cannot find a feasible solution for some instances (two instances at level 4 and one instance at level 5). Notice also that the average computational times increase slightly with the tightness level, especially for the last three algorithms. As a conclusion, we can say that I^2CG_3 remains the best algorithm and that it is more stable against variations of the supplementary constraint right-hand side values.

7 Conclusion

In this paper, we introduced two new versions of the ICG algorithm, called I^2CG_2 and I^2CG_3, which can solve the SPPSC. Because the side constraints in the SPPSC can break the quasi-integrality property of the SPP, the primal algorithms (e.g., ISUD), which pass only from one integer extreme point to an adjacent one, can get stuck in a local minimum. To overcome this limitation, we have added an artificial extreme point which is adjacent to all other extreme points of the feasible region convex hull in a higher dimensional space. Starting from this artificial extreme point, it becomes possible to find an integer direction leading to any integer extreme point of this polyhedron. The difference between algorithms I^2CG_2 and I^2CG_3 is that the latter takes into account the side constraints when searching for a descent direction while the former does not.

The proposed primal algorithms generate a sequence of integer solutions with decreasing costs until an optimal or near-optimal solution is reached. Computational experiments on CPP instances involving up to 1740 set partitioning constraints and 21 side constraints showed that I^2CG_3 slightly outperforms I^2CG_2 and clearly perform better than two popular CG heuristics. For all tested instances, I^2CG_3 finds a large number of integer solutions and reaches a near-optimal solution in less computational time than the other algorithms, on average.

As an extension of this work, we intend to test the proposed algorithm on other types of problems with a variety of side constraints ($\leq$, $=$ and $\geq$) involving variable coefficients that may not be binary.

Appendix A: Detailed sensitivity results

Table 6 reports the sensitivity results for each tested instance and each of the four algorithms I^2CG_2, I^2CG_3, DH, and RMH. The meaning of the columns is the same as for Table 5. For each instance, the best gap and the least computational time are highlighted in bold. Dash (-) entries indicate that the algorithm was unable to find a feasible solution.

Table 6: Sensitivity analysis results of I^2CG_2, I^2CG_3, DH and RMH.

Instance			I^2CG_2						I^2CG_3						*DH*					*RMH*					
Id	*Tasks*	*Level*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*Sc.*	*Cols*	*Time*	*Gap*	*Itr.*	*IntS*	*Sc.*	*Cols*
		1	9	**0.25**	23	87	4	20538	8	**0.25**	22	66	4	20574	11	0.38	76	4	8402	**4**	0.50	20	4	4	8051
		2	9	0.36	23	73	33	23413	8	**0.35**	20	66	42	19848	9	**0.35**	59	33	9474	**5**	0.42	21	5	33	8674
727	239	3	58	0.99	24	110	76	42243	15	**0.73**	25	93	76	28620	**13**	0.89	84	76	10036	14	2.30	24	12	76	8541
		4	15	0.56	23	68	61	28957	13	**0.18**	20	79	61	19614	12	0.63	75	80	9658	**5**	0.50	22	5	76	8718
		5	62	0.87	29	100	80	51434	21	**0.79**	20	64	80	24361	13	1.21	87	76	10543	**9**	1.65	22	7	76	8771
		1	13	**0.16**	16	47	0	25150	10	**0.16**	17	43	4	25589	5	0.36	26	0	10735	**4**	**0.16**	20	3	4	10709
		2	13	0.18	14	56	9	24408	13	**0.16**	19	56	9	28438	5	0.36	26	9	10695	**4**	0.20	17	5	9	10645
09	348	3	45	0.84	19	44	28	44863	17	**0.49**	17	59	42	28462	13	0.83	64	38	12475	**11**	1.09	20	8	33	11553
		4	46	1.09	22	56	61	82220	16	**1.05**	15	55	61	29510	26	1.06	133	80	16582	**11**	1.18	18	11	47	11682
		5	42	1.29	21	51	80	70734	54	**1.04**	20	80	71	42321	**29**	1.27	124	90	16127	60	1.74	25	14	80	13604
		1	26	0.23	19	89	38	51416	31	**0.12**	16	106	38	45874	36	0.18	68	33	17519	**12**	0.14	21	8	33	17010
		2	44	0.21	19	93	66	50823	36	**0.19**	18	101	61	57028	38	0.45	71	66	19615	**15**	0.26	23	5	71	18184
94	424	3	86	0.12	26	104	80	71340	40	**0.06**	18	117	80	54589	36	0.07	68	80	20747	**19**	0.12	29	8	80	19976
		4	175	0.77	46	115	85	322376	87	**0.59**	26	111	90	86525	133	0.68	278	90	35317	**77**	0.88	34	16	85	21198
		5	241	1.87	41	111	85	237759	99	**0.64**	26	131	80	74090	**96**	1.07	210	80	31193	133	0.93	28	16	80	21655
		1	2640	0.17	25	429	19	424760	**1374**	**0.14**	22	469	14	412841	8276	0.32	959	14	164276	7582	0.82	49	43	42	97485
		2	2411	**0.17**	24	455	42	457784	**1669**	**0.17**	23	489	42	419540	8052	0.22	903	52	171415	7640	1.19	53	30	52	103250
95	1255	3	2833	**0.23**	29	516	66	508050	**1893**	**0.23**	24	468	57	467351	9062	1.74	1197	61	192789	7668	2.92	54	100	71	109460
		4	**1748**	0.32	26	482	71	496406	1959	**0.21**	25	486	66	464309	9222	0.44	1088	66	192304	7715	-	53	-	-	111408
		5	**1750**	0.61	25	548	80	550986	3312	**0.22**	31	456	80	608535	10674	0.53	1210	80	205366	7864	-	58	-	-	115258
		1	885	**0.01**	17	322	0	328218	777	**0.01**	16	390	4	353461	3939	0.04	823	0	102596	**562**	0.15	66	8	9	59148
		2	915	**0.01**	17	324	4	326725	**855**	**0.01**	17	402	4	337999	3479	0.03	756	14	105320	1163	0.16	75	8	19	62791
757	1290	3	1042	**0.01**	19	317	19	332758	**827**	**0.01**	17	329	9	323155	3973	**0.01**	844	28	110579	1528	0.05	65	10	33	60859
		4	1106	0.01	23	370	52	398472	**887**	**0.00**	17	378	57	328690	4294	0.04	941	57	122881	7365	0.21	68	11	61	62094
		5	1151	**0.01**	21	388	61	338615	**995**	**0.01**	19	399	76	334044	4577	0.03	1034	76	131420	7362	0.28	67	9	85	65179
		1	1500	0.12	23	440	28	286057	**1286**	**0.09**	22	470	38	286338	4566	0.26	1212	33	130988	7386	0.79	87	18	28	74412
		2	**1482**	0.17	20	488	42	277853	1542	**0.11**	23	462	47	283315	4514	0.24	1170	42	133403	7406	2.14	89	19	76	77687
319	1293	3	1719	0.20	25	391	57	307200	**1661**	**0.09**	25	554	66	303290	4551	0.34	1128	66	132668	7375	1.32	84	25	71	77647
		4	2078	0.36	28	444	71	332187	**2047**	**0.17**	26	547	71	331329	5150	0.56	1385	66	142057	7376	2.23	85	22	71	79578
		5	2059	0.79	28	460	76	371484	2114	**0.15**	27	491	76	332904	5360	1.06	1536	71	147688	7425	-	93	-	-	80754
		1	1711	0.05	20	506	14	372602	1673	**0.04**	18	516	19	361084	8400	0.07	1396	19	150659	**1068**	0.14	112	7	28	89837
		2	2034	0.06	24	438	38	456921	1874	**0.05**	17	565	33	369921	8126	0.08	1317	38	151183	**1144**	0.15	103	13	42	91577
320	1740	3	**1998**	0.06	24	445	57	394734	2076	**0.03**	21	539	52	396851	9122	0.16	1607	66	165260	7818	0.66	106	21	61	92083
		4	**1733**	0.31	22	493	47	376756	2069	**0.03**	22	501	66	377518	9773	0.13	1888	76	175724	7822	0.30	109	7	66	93725
		5	2061	0.30	26	463	76	427415	**2034**	**0.03**	24	548	76	393818	9965	0.21	2053	76	185809	7870	0.50	117	21	76	94633
		Avg.	1044	0.39	24	284	49	254676	**954**	**0.25**	21	305	50	230621	3872	0.47	739	52	92957	3414	0.82	55	15	52	54223

References

[1] Egon Balas and Manfred Padberg. On the set-covering problem. Operations Research, 20(6):1152–1161, 1972.

[2] Egon Balas and Manfred Padberg. On the set-covering problem: Ii. an algorithm for set partitioning. Operations Research, 23(1):74–90, 1975.

[3] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. Operations research, 46(3):316–329, 1998.

[4] Hocine Bouarab, Issmail El Hallaoui, Abdelmoutalib Metrane, and François Soumis. Dynamic constraint and variable aggregation in column generation. European Journal of Operational Research, 262(3):835–850, 2017.

[5] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. Column generation, volume 5. Springer Science & Business Media, 2006.

[6] Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and François Soumis. An improved primal simplex algorithm for degenerate linear programs. INFORMS Journal on Computing, 23(4):569–577, 2011.

[7] Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, and Guy Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming, 123(2):345–370, 2010.

[8] Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. Operations Research, 53(4):632–645, 2005.

[9] Ralph E Gomory. All-integer integer programming algorithm" appearing in industrial scheduling, john f. muth and gerald l. thompson, 1963.

[10] Cédric Joncour, Sophie Michel, Ruslan Sadykov, Dmitry Sverdlov, and François Vanderbeck. Column generation based primal heuristics. Electronic Notes in Discrete Mathematics, 36:695–702, 2010.

[11] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. EURO Journal on Transportation and Logistics, 6(2):111–137, 2017.

[12] Adam N Letchford and Andrea Lodi. Primal cutting plane algorithms revisited. Mathematical Methods of Operations Research, 56(1):67–81, 2002.

[13] Elina Rönnberg and Torbjörn Larsson. Column generation in the integral simplex method. European Journal of Operational Research, 192(1):333–342, 2009.

[14] Elina Rönnberg and Torbjörn Larsson. All-integer column generation for set partitioning: Basic principles and extensions. European Journal of Operational Research, 233(3):529–538, 2014.

[15] Samuel Rosat, Issmail Elhallaoui, François Soumis, and Driss Chakour. Influence of the normalization constraint on the integral simplex using decomposition. Discrete Applied Mathematics, 217:53–70, 2017.

[16] Samuel Rosat, Issmail Elhallaoui, François Soumis, and Andrea Lodi. Integral simplex using decomposition with primal cutting planes. Mathematical Programming, 166(1-2):327–367, 2017.

[17] Adil Tahir, Guy Desaulniers, and Issmail El Hallaoui. Integral column generation for the set partitioning problem. EURO Journal on Transportation and Logistics, pages 1–32, 2019.

[18] Gerald L Thompson. An integral simplex algorithm for solving combinatorial optimization problems. Computational Optimization and Applications, 22(3):351–367, 2002.

[19] V Trubin. On a method of solution of integer linear programming problems of a special kind. In Soviet Mathematics Doklady, volume 10, pages 1544–1546, 1969.

[20] François Vanderbeck. Implementing mixed integer column generation. In Column generation, pages 331–358. Springer, 2005.

[21] Abdelouahab Zaghrouti, Issmail El Hallaoui, and François Soumis. Integral simplex using decomposition for the set partitioning problem. Operations Research, 62(2):435–449, 2014.

[22] Abdelouahab Zaghrouti, Issmail El Hallaoui, and François Soumis. Improved integral simplex using decomposition for the set partitioning problem. EURO Journal on Computational Optimization, 6(2):185–206, 2018.

[23] Abdelouahab Zaghrouti, Issmail El Hallaoui, and François Soumis. Improving set partitioning problem solutions by zooming around an improving direction. Annals of Operations Research, pages 1–27, 2018.