Les Cahiers du GERAD ISSN: 0711–2440

Integral column generation

A. Tahir, G. Desaulniers,
I. Elhallaoui

G–2017–53

June 2017

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

Adil Tahir
Guy Desaulniers
Issmail Elhallaoui

GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7

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

June 2017

Les Cahiers du GERAD
G–2017–53

Abstract: The integral simplex using decomposition (ISUD) algorithm was recently developed to solve efficiently set partitioning problems containing a number of variables that can all be enumerated a priori. This primal algorithm generates a sequence of integer solutions with decreasing costs, leading to an optimal or near-optimal solution depending on the stopping criterion used. In this paper, we develop an integral column generation (ICG) heuristic that combines ISUD and column generation to solve set partitioning problems with a very large number of variables. Computational experiments on instances of the public transit vehicle and crew scheduling problem and of the airline crew pairing problem involving up to 2000 constraints show that ICG clearly outperforms two popular column generation heuristics (the restricted master heuristic and the diving heuristic). ICG can yield optimal or near-optimal solutions in less than one hour of computational time, generating up to 300 integer solutions during the solution process.

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

1 Introduction

Many complex industrial problems are formulated as a set partitioning problem (SPP). These problems include, among others, the integrated vehicle and crew scheduling problem arising in public transit (VCSP, see Haase et al. [10]) and the airline crew pairing problem (CPP, see Desaulniers et al. [6]) which are often solved by branch-and-price (see Barnhart et al. [3]). The SPP is a combinatorial optimization problem that can be defined as follows. Let $T = \{1, \ldots, m\}$ be a set of tasks to accomplish exactly once (e.g., flights to operate, customers to visit once each, etc.). Let $N = \{1, \ldots, n\}$ be a set of feasible task subsets (e.g., defined by crew schedules, vehicle routes, etc.). For each task subset $j \in N$, let c_j be the least cost to accomplish these tasks and let a_{ij}, $i \in T$, be a binary parameter indicating if task i is in subset j or not. The SPP consists of selecting subsets in N such that each task in T belongs to exactly one of the selected subset and the sum of the costs of these subsets is minimized. It can be formulated as the following integer program:

$$\min_{x} \quad \sum_{j \in N} c_j x_j \tag{1}$$

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

$$x_j \in \{0, 1\}, \qquad \forall j \in N, \tag{3}$$

where x_j, $j \in N$, is a binary variable equal to 1 if subset j is selected and 0 otherwise. The objective function (1) aims at minimizing the total cost. The set partitioning constraints (2) ensure that each task is included in a single selected subset. Finally, binary requirements on the x_j variables are expressed by (3). Note that additional (non-set-partitioning) constraints can be considered to model other aspects of the problem at hand such as vehicle availability in vehicle scheduling problems. In this work, we focus on the pure SPP, i.e., without additional constraints.

In the following, we denote by H the convex hull of the set of the feasible solutions of (1)–(3) and by R the feasible region of the linear relaxation of (1)–(3). Notice that H and R are convex polytopes if they are non-empty.

Since 1969, researchers have been attracted by the SPP because its polytope R possesses the *quasi-integrality* property. This property, which distinguishes the SPP from most of the other problems, stipulates that every edge of H is also an edge of R. This implies that every integer solution corresponding to an extreme point of H is also an extreme point of R and, for every pair of such integer points, there exists on the boundary of R a path linking them which is composed of edges linking only integer extreme points. The study of this property has led to the development of several interesting solution algorithms (see Balas and Padberg [2], Thompson [22], and Saxena [21]), including the Integral Simplex Using Decomposition (ISUD) algorithm of Zaghrouti et al. [25]. These algorithms will be reviewed in the next section.

This paper continues these works on the SPP. Indeed, we exploit existing theoretical results to propose a new Integral Column Generation (ICG) algorithm. Starting from a possibly poor-quality solution that might even be infeasible, this new primal method finds a sequence of improving feasible integer solutions. At the opposite of the branch-and-price algorithm which only uses dual information to generate new variables, the ICG algorithm exploits both primal and dual information to do so. On the one hand, the current dual solution is used to find new columns (task subsets) that can potentially be part of an improved integer solution. On the other hand, the primal information corresponding to the current integer solution is exploited to find descent directions yielding better integer solutions. To increase the number of descent directions found at each iteration of the ICG algorithm, we have chosen to generate a very large number of columns each time that the column generation subproblem is solved. In a traditional column generation algorithm, this strategy is avoided as it typically increases substantially the time to re-optimize the restricted master problem (RMP) and, therefore, the total computational time. However, in the ICG algorithm, it is rather beneficial as it increases the probability of finding several improved integer solutions. Furthermore, it does not bother the solution of the RMP because this one is decomposed and the density of the constraint coefficient matrix of the original problem is reduced as explained in Section 3.

Although the proposed ICG algorithm could be designed to be exact, our implementation is heuristic. It has been tested on instances of the VCSP and of the CPP, with up to 2000 and 1740 constraints in the master problem, respectively. Our computational results show the effectiveness of the proposed algorithm compared to two popular column generation heuristics often applied for solving large instances of these problem types. Indeed, we succeed to compute a better quality solution (often optimal or near-optimal) in much less computational time for most instances. This is achieved by performing a small number of column generation iterations (at most 12 for the VCSP and 23 for the CPP) and finding a large number of integer solutions throughout the solution process (around 10 solutions per column generation iteration on average).

This paper is structured as follows. In the next section, we review the literature on the SPP and on the primal and dual-fractional algorithms that are the most commonly used for solving this problem. In Section 3, we present the ISUD algorithm and related concepts which are at the basis of the ICG algorithm. Section 4 describes this new algorithm. Section 5 reports the results of our computational experiments. Finally, conclusions are drawn in Section 6.

2 Literature review

In the literature on the SPP, there are two main types of solution algorithms, namely, dual-fractional and primal algorithms. Primal algorithms move from one feasible (integer) solution to another, whereas dual-fractional algorithms allow infeasible (non-integer) solutions. In this review, we focus in Section 2.1 on a single family of dual-fractional algorithms, the branch-and-price (BP) algorithms which can handle SPP instances with a very large number of variables, and in Section 2.2 on two types of primal algorithms, namely, the integral simplex algorithms and the ISUD algorithm.

2.1 Branch-and-price algorithms

Branch-and-price (see Barnhart et al. [3]) is a popular and efficient solution method for the SPP. It embeds column generation (CG) in a branch-and-bound framework, where a linear relaxation of SPP is solved using CG at each node of the search tree. CG is designed to solve linear programs containing a very large number of variables that are often obtained through Dantzig-Wolfe decomposition (Dantzig and Wolfe [5]). In this context, the linear program is called the *master problem* (MP) and the CG algorithm solves at each iteration the MP restricted to a subset of the variables, called the *RMP*, and one or several *subproblems*. In our case, the MP corresponds to the linear relaxation of (1)–(3) and the subproblem is often modeled as a shortest path problem with resource constraints (SPPRC, see [12]). Solving the RMP provides a primal solution but also a dual solution denoted $\boldsymbol{\alpha} \in \mathbb{R}^m$. The subproblem aims at finding new columns associated with variables x_j of negative reduced cost $\bar{c}_j = c_j - \sum_{i \in T} \alpha_i a_{ij}$ that are added to the RMP before starting a new iteration. If no such columns can be generated, then the CG algorithm stops and the optimal solution of the current RMP is also optimal for the MP (setting all unknown variables to 0).

It is well known that the standard CG algorithm may be subject to several convergence issues (see Vanderbeck [24]) which transfer to the branch-and-price algorithm. In the literature, several dual variable stabilization techniques have been introduced to improve convergence (see, e.g., [15, 24]). In 2005, Elhallaoui et al. [8] have proposed the dynamic constraint aggregation (DCA) algorithm for solving the SPP. This algorithm consists of reducing the number of set partitioning constraints by aggregating some of them as needed. The aggregation is revised during the solution process to ensure the exactness of the DCA algorithm. The authors report computational results obtained on randomly generated VCSP instances which show that DCA clearly outperforms standard CG for solving the linear relaxation of the SPP: computational time reductions of up to 80% were achieved. An improved variant of the DCA algorithm called the multi-phase DCA (MPDCA) algorithm was also developed by Elhallaoui et al. [9]. This variant allows a further reduction of degeneracy and of the number of fractional-valued variables in the computed MP solutions. Consequently, it often finds integer solutions. More recently, to alleviate the drawbacks of the standard CG algorithm, Bouarab et al. [4] proposed three new decompositions that integrates the improved primal simplex (IPS) algorithm of Elhallaoui et al. [7] and either CG or DCA. The best decomposition combines IPS and DCA and reduces the number of non-zero elements in the constraint coefficient matrix. This opens up the possibility

to consider more columns in the RMP and, thus, to stabilize the dual variables. The computational results reported by Bouarab et al. [4] show that their IPS/DCA algorithm can be eight times faster than the standard CG algorithm on the instances tested by Elhallaoui et al. [8, 9]. These results also show that the application of a clever decomposition in a CG framework is a promising research avenue that is worth exploring to overcome the convergence issues associated with standard CG.

2.2 Primal algorithms

Here, we review first the integral simplex algorithms without decomposition before discussing the ISUD algorithm.

2.2.1 Integral simplex algorithms without decomposition

In 1969, Trubin [23] proved that the SPP polytope has the quasi-integrality property: every edge of H is an edge of R. This property implies that, for every pair of feasible solutions, there exists a path between them that is composed of edges of R, visits only integer solutions at the edge extremities, and the sequence of the costs of the visited solutions is decreasing (see Balas and Padberg [1]). These authors also showed that an optimal solution can be reached in at most m pivots from any initial feasible solution. This result is, however, impractical because it requires the knowledge of this optimal solution. Based on these results, Thompson [22] introduced in 2002 the integral simplex algorithm, which proceeds in two phases. The first phase (local method) consists of performing, as long as possible, non-degenerate pivots to reach a local optimum. All these pivots are performed on a coefficient (that of the entering variable in the row associated with the leaving variable) equal to 1. The second phase (global method) builds a tree, where each node corresponds to a subproblem that is solved using the local method. An optimal solution to SPP is given by the best solution found in the nodes explored in the tree. Soon after, Saxena [21] proposed an improvement to Thompson's algorithm that allows pivoting on a -1 coefficient and relies on anti-cycling rules. The great advantage of these integral simplex algorithms is that they always have a feasible integer solution at hand. However, these algorithms suffer from degeneracy and might struggle to find a non-degenerate pivot.

Rönnberg and Larsson [16, 17] have developed combined CG/integral simplex algorithms, called all-integer column generation algorithms. They first adapted the necessary and sufficient condition on the columns yielding non-degenerate pivots proposed by Balas and Padberg [1] and derived from it a condition to identify columns yielding degenerate pivots. Depending on the type of columns to be generated, one of these conditions is integrated in the column generation subproblem to restrict the set of columns that can be generated. The local method of the integral simplex algorithm iterates between performing non-degenerate pivots in the RMP, generating columns yielding non-degenerate pivots, performing degenerate pivots in the RMP, and generating columns yielding degenerate pivots, until reaching a stopping criterion. Then, the global method of the integral simplex algorithm which explores a search tree is applied to complete the solution process. The authors have described academic examples to illustrate the unfolding of this algorithm but no extensive computational experiments were conducted. Tests on small-sized instances of the generalized assignment problem show that degeneracy remains an issue.

Note that these algorithms generate sequences of improving integer solutions. To move from one solution to the next, they perform a (possibly empty) sequence of degenerate pivots followed by a non-degenerate pivot. The selection of the degenerate pivots in each sequence is somewhat arbitrary and, thus, may yield long sequences and waste substantial computational time.

2.2.2 The ISUD algorithm

In 2011, Elhallaoui et al. [7] introduced the improved primal simplex (IPS) algorithm for solving efficiently linear programs subject to high degeneracy. The success of this algorithm has motivated an attempt to adapt it to an integer program, namely, the SPP. In 2014, Zaghrouti et al. [25] realized this IPS adaptation by introducing a new algorithm called the ISUD algorithm. As in the IPS algorithm, the ISUD algorithm relies on a decomposition concept specialized for degenerate problems. Indeed, it decomposes the SPP in a reduced problem (RP) and a complementary problem (CP) that searches for descent directions that can

improve the current integer solution. The definitions of RP and CP as well as the ISUD algorithm will be given in Section 3. Compared to the algorithms described in Section 2.2.1, the ISUD algorithm finds at each iteration a combination of columns to pivot into the basis. This combination ensures a decrease in the objective function and is minimal in the sense that, if one of the columns it contains is not pivoted into the basis, then no improvement can be achieved. The computational experiments performed by Zaghrouti et al. [25] on instances of the VCSP and of the CPP involving up to 1,600 constraints and 500,000 variables showed the effectiveness of the ISUD algorithm to find optimal or near-optimal solutions much more rapidly than the CPLEX mixed-integer programming (MIP) solver for the 90 tested instances. Note, however, that, in these instances, all columns are known a priori.

To improve the ISUD algorithm, further research has been realized by Rosat et al. [18, 19] and Zaghrouti et al. [26]. These works focused on a weakness of the ISUD algorithm, namely, solving the CP may sometimes require some kind of branching to ensure finding a descent direction leading to an integer solution, called hereafter an *integer direction*. Rosat et al. [19] have proposed cuts to avoid branching. These cuts are, however, costly to separate and several of them might be needed to find an integer direction. Alternatively, Rosat et al. [18] studied the role of the normalization constraint added to the CP to bound it. The left-hand side of this constraint is defined as a weighted sum of variables. The authors showed the existence of a weight vector that enables finding only integer directions until reaching optimality. Unfortunately, no procedure for building this vector is proposed. Finally, Zaghrouti et al. [26] developed a variant of the ISUD algorithm called the zooming algorithm. When it is not possible to find an integer direction without branching, the CP provides a non-integer descent direction that can be used to define a neighborhood around the current integer solution. This neighborhood is of small size and can, thus, be explored efficiently using a MIP solver to find an integer direction.

From this literature review, we observe that the best primal algorithm for the SPP, namely, the ISUD algorithm, has not been adapted to the CG context. Consequently, this paper constitutes the first attempt at combining CG and ISUD to compute efficiently sequences of improving integer solutions for large-scale SPP instances.

3 Preliminaries

In this section, we provide preliminaries that are needed to understand the proposed ICG algorithm. They include a description of the ISUD algorithm of Zaghrouti et al. [25]. In the following, we use bold characters to denote vectors and matrices.

3.1 Description of the ISUD algorithm

Let $\mathbf{x}$ be a feasible solution to SPP (1)–(3) and $S = \{j \mid x_j = 1\}$ the set of the indices of the task subsets selected in this solution. Let $\mathbf{A} = (a_{ij})_{i \in T, j \in N}$ be the constraint coefficient matrix and denote by $\mathbf{A}_j$ its column associated with variable x_j, $j \in N$. By breach of terminology, we say that S is the solution $\mathbf{x}$ and that the columns $\mathbf{A}_j$, $j \in S$, are in the solution $\mathbf{x}$ or in S. The other columns $\mathbf{A}_j$, $j \in N \setminus S$, are either *compatible* or *incompatible* with the columns in S according to the following definition.

Definition 1 *An arbitrary column in $\{0,1\}^m$ (not necessarily a column of $\mathbf{A}$) is said to be compatible with the columns in S (or, simply, compatible with S) if it can be written as a linear combination of these columns.*

Let $\mathbf{A}_U = \left(\mathbf{A}_j\right)_{j \in U}$ be a submatrix of $\mathbf{A}$. We denote by $\mathbf{A}_U^1$ the submatrix of $\mathbf{A}_U$ composed of its first $|U|$ linearly independent rows and all its columns and by $\mathbf{A}_U^2$ the submatrix of $\mathbf{A}_U$ containing the remaining rows. Furthermore, let C_S be the index set of the columns $\mathbf{A}_j$, $j \in N$, that are compatible with S and let $I_S = N \setminus C_S$ be the index set of the incompatible columns. Observe that $S \subseteq C_S$ and that $\mathbf{A}_{C_S}^1$ contains $|S|$ rows, namely, one for the first task covered by each column $\mathbf{A}_j$, $j \in S$. As mentioned in the previous section, the SPP is decomposed in two problems: a RP built from the compatible columns and a CP containing the incompatible ones.

The RP is given by:

$$\min_{\mathbf{x}_{C_S}} \quad \mathbf{c}_{C_S}^\top \mathbf{x}_{C_S} \tag{4}$$

$$\text{s.t.:} \quad \mathbf{A}^1_{C_S} \mathbf{x}_{C_S} = \mathbf{e} \tag{5}$$

$$\mathbf{x}_{C_S} \in \{0,1\}^{|C_S|} \tag{6}$$

where $\mathbf{x}_{C_S}$ is the subvector of the variables associated with the columns compatible with S, $\mathbf{c}_{C_S}$ is the subvector of the cost coefficients associated with these variables, and $\mathbf{e} \in \{1\}^{|S|}$ is a vector of ones. In RP, the current solution S is associated with a basis corresponding to the identity matrix. Consequently, pivoting into the basis any compatible column that has a negative reduced cost results in an improved integer solution S' (the entering column replaces two or more columns in the solution). This first pivot is, thus, non-degenerate. One can then redefine RP with respect to the new solution S' and search for a new negative reduced cost column in $C_{S'}$ that is compatible with S'. This process can be repeated until no such column is found.

The incompatible columns $\mathbf{A}_j$, $j \in I_S$, are used in the CP to find an integer descent direction. This direction is obtained by finding a linear combination of the incompatible columns that is compatible with S and composed of disjoint columns according to the following definition.

Definition 2 *Let $U \subseteq N$ be a subset of column indices. The columns $\mathbf{A}_j$, $j \in U$, are said to be* disjoint *if and only if $\mathbf{A}_{j_1}^\top \mathbf{A}_{j_2} = 0$ for all $j_1, j_2 \in U$ such that $j_1 \neq j_2$.*

Let v_j, $j \in I_S$, be the weight variables defining the linear combination of the incompatible columns; λ_l, $l \in S$, the weight variables defining the linear combination of the columns in the solution S; and w_j, $j \in I_S$, nonnegative weights used in the normalization constraint. Furthermore, denote by $F_S = \{(j_1, j_2) \in I_S \times I_S \mid \mathbf{A}_{j_1}^\top \mathbf{A}_{j_2} \neq 0\}$ the set of index pairs of non-disjoint incompatible columns. The CP can be formulated as follows:

$$z^{CP} = \min_{v,\lambda} \quad \sum_{j \in I_S} c_j v_j - \sum_{l \in S} c_l \lambda_l \tag{7}$$

$$\text{s.t.:} \quad \sum_{j \in I_S} v_j \mathbf{A}_j - \sum_{l \in S} \lambda_l \mathbf{A}_l = 0 \tag{8}$$

$$\sum_{j \in I_S} w_j v_j = 1 \tag{9}$$

$$v_j \geq 0, \qquad \forall j \in I_S \tag{10}$$

$$v_{j_1} v_{j_2} = 0, \qquad \forall (j_1, j_2) \in F_S. \tag{11}$$

The objective function (7) searches for a direction that will incur a cost decrease. Constraint (8) ensures that the linear combination of the incompatible columns is compatible with S. The normalization constraint (9) simply bounds the feasible region. Nonnegativity constraints on the v_j variables are enforced through (10). Given these nonnegativity requirements and the facts that the elements of $\mathbf{A}$ are binary and the columns in S are disjoint, the λ_l variables are also nonnegative. Finally, the disjunctive constraints (11) impose the selection of disjoint columns in the linear combination of the incompatible variables, ensuring an integer direction.

If $z^{CP} < 0$, then an integer descent direction $\mathbf{d} = \left(d_j\right)_{j \in N}$ is identified by the positive-valued v_j and λ_l variables as follows:

$$d_j = \begin{cases} 1 & \text{if } j \in I_S \text{ and } v_j > 0 \\ -1 & \text{if } j \in S \text{ and } \lambda_j > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Fixing the step length to 1, an improved integer solution is obtained by replacing the columns $\mathbf{A}_l$, $l \in S$, with $\lambda_l > 0$ by the incompatible columns $\mathbf{A}_j$, $j \in I_S$, with $v_j > 0$. This corresponds to entering the latter columns into the basis, while removing the former from it. On the other hand, if $z^{CP} \geq 0$, then no integer descent direction can be found and solution S is, therefore, optimal.

In practice, constraints (11) are relaxed from the initial CP, yielding the *relaxed CP* which can then be solved by a linear programming solver. As empirically shown by Zaghrouti et al. [25], the selected columns in the linear combination of incompatible columns are often disjoint and the computed solution is thus feasible with respect to the relaxed constraints (11). When this is not the case, branching can be performed to enforce these constraints. For example, for each column $\mathbf{A}_j$, $j \in I_S$, with $v_j > 0$ in the solution of the relaxed CP, one can create a child node that imposes $v_j = 0$.

The main steps of the ISUD algorithm described above are summarized in Algorithm 1. The performance of this algorithm mainly depends on the effectiveness at solving the CP in Step 3. Below, we discuss different avenues that have been studied to ease its solution.

Algorithm 1 ISUD algorithm

1: Start from an initial solution S
2: Improve the current solution S through the RP
3: Solve the CP
4: **if** $z^{CP} < 0$ **then**
5: Update solution S according to the integer direction found
6: Go to Step 2
7: Return the current solution S

3.2 Compatibility matrices

Observe that the density of the constraint coefficient matrix in the relaxed CP is very similar to that of the original SPP. To reduce this density, one can perform some variable substitution. Here, we present a first reformulation (proposed by Zaghrouti et al. [25]) that is obtained by isolating each λ_l variable in the last constraint (8) in which it appears before substituting it in the rest of the formulation. These substitutions yield the following reformulation of the CP:

$$\min_{v} \quad (\mathbf{c}_{I_S}^\top - \mathbf{c}_S^\top (\mathbf{A}_S^1)^{-1} \mathbf{A}_{I_S}^1)^\top \mathbf{v} \tag{13}$$

$$\text{s.t.:} \quad (\mathbf{A}_S^2 (\mathbf{A}_S^1)^{-1} \mathbf{A}_{I_S}^1 - \mathbf{A}_{I_S}^2)\mathbf{v} = 0 \tag{14}$$

$$\sum_{j \in I_S} w_j v_j = 1 \tag{15}$$

$$v_j \geq 0, \qquad \forall j \in I_S \tag{16}$$

$$v_{j_1} v_{j_2} = 0, \qquad \forall (j_1, j_2) \in F_S, \tag{17}$$

where, for $U \subset N$, $\mathbf{c}_U$ is the subvector of the cost coefficient vector in (7) associated with the variables v_j, $j \in U$. Constraint (14) can be written in the form $(\mathbf{M}_1 \mathbf{A}_{I_S})^\top \mathbf{v} = 0$, where $\mathbf{M}_1$ is a compatibility matrix according to the following definition.

Definition 3 *A matrix* $\mathbf{M}$ *is said to be a* compatibility matrix *if and only if* $\mathbf{M}\mathbf{A}_j = 0$ *for all compatible columns* $\mathbf{A}_j$, $j \in C_S$, *and* $\mathbf{M}\mathbf{A}_j \neq 0$ *for all incompatible columns* $\mathbf{A}_j$, $j \in I_S$.

There exists an infinite number of compatibility matrices that can be used to define the CP. The choice of this matrix may have a significant impact on the computational times. We have chosen to use the matrix $\mathbf{M}_2$ introduced by Bouarab et al. [4] that is specialized for vehicle routing and crew scheduling problems. In these problems, each column is associated with a route or a crew schedule that performs a subset of tasks in a given sequence. The $\mathbf{M}_2$ matrix allows to measure the deviation of the incompatible columns with respect to the task sequences defined by the columns in solution S. This matrix is such that, if an incompatible column $\mathbf{A}_j$, $j \in I_S$, does not respect the task ordering in a sequence, $\mathbf{M}_2\mathbf{A}_j$ contains a component equal to -1 each time that $\mathbf{A}_j$ covers a task but not its predecessor and a component equal to 1 each time that $\mathbf{A}_j$ covers a task but not its successor (see Bouarab et al. [4] for details).

3.3 Multi-phase strategy and normalization constraint weights

The usage of matrix $\mathbf{M}_2$ is combined with a partial pricing strategy called the *multi-phase* strategy. Each time that the CP needs to be solved, a sequence of phases can be invoked. Each phase is defined by a

parameter k (a positive integer) which restricts the CP to a subset $I_S^k \subseteq I_S$ of the incompatible columns such that $I_S^k \subseteq I_S^\ell$ if $k < \ell$. The sequence of phases is predetermined and corresponds to an increasing sequence $k_1, k_2, ..., k_p$ of values of k with $k_p = \infty$ (for instance, $k = 2, 3, 4, 5, 6, \infty$), where phase $k = k_p = \infty$ means that $I_S^\infty = I_S$. Starting in phase k_1, the CP restricted to the subset $I_S^{k_1}$ is solved. If its optimal value is negative, the process stops and the computed linear combination of incompatible columns is returned. Otherwise, the next phase is invoked. The process repeats until obtaining a negative optimal value for the CP or reaching phase $k_p = \infty$.

The definition of a subset I_S^k used in phase k is based on a distance between an incompatible column and the vector subspace generated by the columns in the solution. This distance, called the degree of incompatibility, is defined as follows.

Definition 4 *The* degree of incompatibility δ_j *of an incompatible column* $\mathbf{A}_j$, $j \in I_S$, *is given by* $\delta_j = ||\mathbf{M}\mathbf{A}_j||$, *where* $\mathbf{M}$ *is a compatibility matrix.*

In phase k of the multi-phase strategy for solving the CP, the index subset of the incompatible columns considered in the CP is defined by $I_S^k = \{j \in I_S \,|\, \delta_j \leq k\}$. Bouarab et al. [4] has proven the following result.

Proposition 1 *In phase* k, *each incompatible column* $\mathbf{A}_j$, $j \in I_S^k$, *has at most* $k+1$ *non-zero coefficients in the constraint matrix of the CP if* $\mathbf{M}_2$ *is used as the compatibility matrix.*

Consequently, for phases with a low k value, the constraint coefficient matrix has a low density. This helps to reduce the computational effort for solving the CP and also to produce integer directions without branching. Finally, it may allow to consider a very large number of incompatible columns in the CP if many are available.

For our tests, we have chosen to set $w_j = \delta_j$ for all $j \in I_S$. Rosat et al. [18] showed that using these weights in the normalization constraint (9) favors finding integer directions in the relaxed CP.

3.4 Computation of a complete dual solution

In a column generation context, a dual solution $\boldsymbol{\alpha} = \left(\alpha_i\right)_{i \in T} \in \mathbb{R}^m$ to the MP (i.e., the linear relaxation of (1)–(3)) is required at each column generation iteration to define the subproblem objective function and generate new negative reduced cost columns. Such a complete solution is not directly available in ISUD. However, to build such a solution, we use as in Bouarab et al. [4] the dual variable values $\hat{\boldsymbol{\pi}} \in \mathbb{R}^{m-|S|}$ associated with constraints (14) that were computed when solving the last relaxed CP (13)–(16) (i.e., without constraints (17)). Recall that, for each column index $j \in S$, there are $|T_j| - 1$ constraints (14) in the CP, where T_j is the subset of tasks covered by $\mathbf{A}_j$. These constraints are associated with the first $|T_j| - 1$ tasks of T_j. We denote by $\hat{\pi}_j^q$, for all $j \in S$ and $q \in \{1, \ldots, |T_j| - 1\}$, the component of $\hat{\boldsymbol{\pi}}$ associated with the task q covered by $\mathbf{A}_j$. Similarly, we denote by α_j^q, for all $j \in S$ and $q \in \{1, \ldots, |T_j|\}$, the component of $\boldsymbol{\alpha}$ associated with the task q covered by $\mathbf{A}_j$. To determine values for the α_j^q variables, we solve the following linear system of equations:

$$\sum_{i=1}^{|T_j|} \alpha_j^i = c_j, \qquad \forall j \in S, \tag{18}$$

$$\sum_{i=1}^{q} \alpha_j^i = \hat{\pi}_j^q, \qquad \forall j \in S,\ q \in \{1, \ldots, |T_j| - 1\} \tag{19}$$

which can be decomposed by column index $j \in S$. Conditions (18) ensure that the columns in the current solution of the SPP have a zero reduced cost and all compatible columns in the RP have, thus, a nonnegative reduced cost. Furthermore, conditions (19) ensure that the least value among all weighted reduced costs $\bar{c}_j / w_j = (c_j - \boldsymbol{\alpha}^\top \mathbf{A}_j)/w_j$, $j \in I_S$, of the incompatible columns considered in the CP is maximized.

The following definition will be useful for the next section.

Definition 5 *A dual solution* $\boldsymbol{\alpha} \in \mathbb{R}^m$ *that satisfies conditions (18) is said to* correspond to the current solution S.

Finally, observe that, if $(\hat{\boldsymbol{\pi}}, \hat{\sigma}) \in \mathbb{R}^{m-|S|} \times \mathbb{R}$ is an optimal dual solution to the relaxed CP (13)–(16), then $(\boldsymbol{\alpha}, \hat{\sigma}) \in \mathbb{R}^m \times \mathbb{R}$ forms an optimal solution to the relaxed CP (7)–(10) when $\boldsymbol{\alpha}$ is computed by solving the equation system (18)–(19). In these solutions, $\hat{\sigma}$ is the dual value associated with constraints (9) and (15), which is equal to the optimal value of the relaxed CP.

4 Methodology

In this section, we start by describing the ICG algorithm before presenting an acceleration strategy. In the rest of the text, the words *variable* and *column* are used interchangeably.

4.1 The ICG algorithm

The ICG algorithm is based on the three-level decomposition illustrated in Figure 1. The first two levels correspond to the RMP in a column generation algorithm except that this RMP is an integer linear program, not a continuous linear program. The RMP is decomposed into a RP and a CP, and solved using the ISUD algorithm. The third level of the decomposition contains the column generation subproblem which generates negative reduced cost columns that are added to the pool of columns available for the RMP. In our tests, the subproblem can be separated into several subproblems. These subproblems are SPPRCs that are solved by a labeling algorithm.

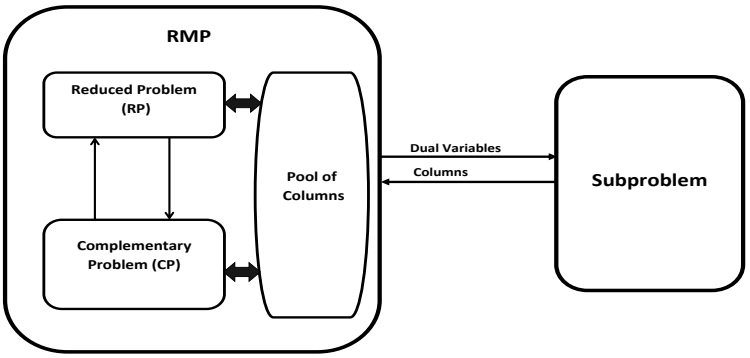

Figure 1: The three-level decomposition of the ICG algorithm

A pseudo-code of the ICG algorithm is given in Algorithm 2. It starts by computing an initial primal solution and an initial dual solution. In Section 5, we will describe how these solutions are computed for our test problems. It then initializes an iteration counter r, a current upper bound ℓ on the degree of incompatibility, and the number $consFail$ of consecutive iterations in which the cost improvement (compared to the previous iteration) is not considered sufficient. As in the ISUD algorithm, the ICG algorithm applies a multi-phase strategy when generating the columns. In fact, the subproblem includes an additional constraint which imposes that the degree of incompatibility of the feasible columns be less than or equal to ℓ. Note that this multi-phase strategy is independent of the one used in the ISUD algorithm when solving the RMP (Steps 12 and 17).

The main loop (Steps 3–21) starts by solving the subproblem denoted $SP(\boldsymbol{\alpha}^r, \ell)$ taking into account the current dual solution $\boldsymbol{\alpha}^r$ and the current limit on the degree of incompatibility ℓ. The negative reduced cost columns generated by the subproblem, if any, are stored in set N'. If no columns are generated, ℓ is increased by one and the subproblem is solved again. This process is repeated until finding negative reduced cost columns or until ℓ exceeds a predefined value $maxDegree$. In the latter case, the algorithm stops and outputs (in Step 9) the current solution x^r as the best solution found. Note that, in practice, we try to generate a large number of columns at each iteration. Given that a multi-phase strategy is used in ISUD when solving the RMP, this large number of columns does not hinder the solution process.

When columns are found ($N' \neq \emptyset$), they are added to the pool of columns N available to the RMP. The RMP is then solved by the ISUD algorithm which returns the current solution $\mathbf{x}^r$, its cost z^r and a dual

Algorithm 2 ICG algorithm

1: Find an initial primal solution $\mathbf{x}^0$ and an initial dual solution $\boldsymbol{\alpha}^0$
2: $r \leftarrow 0$; $\ell \leftarrow 1$; $consFail \leftarrow 0$
3: **while** $consFail \leq maxConsFail$ **do**
4: **repeat**
5: $N' \leftarrow$ Solve $SP(\boldsymbol{\alpha}^r, \ell)$
6: **if** $N' = \emptyset$ **then**
7: $\ell \leftarrow \ell + 1$
8: **if** $\ell > maxDegree$ **then**
9: Return $\mathbf{x}^r$
10: **until** $N' \neq \emptyset$
11: $N \leftarrow N \cup N'$; $r \leftarrow r + 1$
12: $(z^r, \mathbf{x}^r, \boldsymbol{\alpha}^r) \leftarrow$ Solve $RMP(\mathbf{x}^r)$ using ISUD
13: **if** $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$ **then**
14: $(z^r, \bar{\mathbf{x}}^r) \leftarrow$ Solve $MIP(\mathbf{x}^r)$
15: **if** $\bar{\mathbf{x}}^r \neq \mathbf{x}^r$ **then**
16: $\mathbf{x}^r \leftarrow \bar{\mathbf{x}}^r$
17: $(z^r, \mathbf{x}^r, \boldsymbol{\alpha}^r) \leftarrow$ Solve $RMP(\mathbf{x}^r)$ using ISUD
18: **if** $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$ **then**
19: $consFail \leftarrow consFail + 1$; $\ell \leftarrow \ell + 1$
20: **else**
21: $consFail \leftarrow 0$
22: Return $\mathbf{x}^r$

solution $\boldsymbol{\alpha}^r$ corresponding to this current solution. This dual solution is computed as described in Section 3.4. In the ISUD algorithm, a multi-phase strategy is applied and a relaxed CP (without constraints (17)) is always solved, i.e., branching is never performed to obtain an integer solution. In practice, the ISUD algorithm often terminates because the direction found when solving the last relaxed CP is not integer.

If the cost of the solution returned by ISUD does not yield a sufficiently large improvement compared to the cost achieved at the previous iteration, i.e., if $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$, where $minImp$ is a predefined parameter value, then a small-sized mixed integer program, denoted $MIP(\mathbf{x}^r)$, is solved by a commercial MIP solver in Step 14. This program is defined as the current RP augmented by all columns in N that have a relatively small degree of incompatibility (e.g., smaller than 7). Notice that solving this program can yield an improved solution only if the ISUD algorithm stops with a non-integer direction in Step 12. Otherwise, solving $MIP(\mathbf{x}^r)$ is useless and can be omitted. If the computed solution $\bar{\mathbf{x}}^r$ of $MIP(\mathbf{x}^r)$ differs from the current solution $\mathbf{x}^r$, then $\bar{\mathbf{x}}^r$ becomes the current solution and ISUD is applied again in Step 17 to try to improve this new solution considering all columns in N and not only a subset of it like in $MIP(\mathbf{x}^r)$. The cost improvement realized in this iteration is again checked in Step 18. If this improvement is deemed insufficient, the number of consecutive iterations where the algorithm failed to yield a sufficiently large cost improvement is incremented by one, as well as the upper bound on the degree of incompatibility ℓ, before starting a new iteration or ending the whole solution process if $consFail$ becomes equal to $maxConsFail$, a predefined parameter value. Otherwise, the number of consecutive failures is reset to zero.

The ICG Algorithm 2 is heuristic for two reasons. First, the stopping criterion may be too restrictive. The parameters $maxDegree$ and $maxConsFail$ should be set to large unrestrictive values in an exact algorithm. Second, the disjunctive constraints (17) are always relaxed from the CP and solving it might not return an integer descent direction even if one exists. Consequently, in an exact algorithm, these constraints should be considered by the algorithm solving the CP.

Note that the procedure used to compute a complete dual solution $\boldsymbol{\alpha}$ is valid to ensure the exactness of the ICG algorithm. Indeed, as shown by the next proposition, at least one variable x_j has a negative reduced cost with respect to $\boldsymbol{\alpha}$ if the current solution is not optimal.

Proposition 2 *Let $\boldsymbol{\alpha} \in \mathbb{R}^m$ be a dual solution corresponding to a solution S with cost $\sum_{j \in S} c_j$. Let S' be an improved solution ($\sum_{j \in S'} c_j < \sum_{j \in S} c_j$). Then, there exists at least one variable x_j, $j \in S' \setminus S$, that has a negative reduced cost $c_j - \sum_{i \in T} a_{ij} \alpha_i$ with respect to the dual solution $\boldsymbol{\alpha}$.*

Proof (by contradiction). Assume that all variables x_j, $j \in S' \setminus S$, have a non-negative reduced cost. In this case, we have

$$c_j \geq \sum_{i \in T} a_{ij}\alpha_i, \qquad \forall j \in S' \setminus S \quad \Rightarrow \quad \sum_{j \in S' \setminus S} c_j \geq \sum_{i \in T} \sum_{j \in S' \setminus S} a_{ij}\alpha_i \tag{20}$$

$$c_j = \sum_{i \in T} a_{ij}\alpha_i, \qquad \forall j \in S \setminus S' \quad \Rightarrow \quad \sum_{j \in S \setminus S'} c_j = \sum_{i \in T} \sum_{j \in S \setminus S'} a_{ij}\alpha_i. \tag{21}$$

Given that the sum of the columns in $S' \setminus S$ is compatible with the columns in $S \setminus S'$, it ensues that

$$\sum_{i \in T} \sum_{j \in S' \setminus S} a_{ij}\alpha_i = \sum_{i \in T} \sum_{j \in S \setminus S'} a_{ij}\alpha_i. \tag{22}$$

From (20)–(22), we deduce that $\sum_{j \in S' \setminus S} c_j \geq \sum_{j \in S \setminus S'} c_j$, which contradicts the fact that S' is a better solution than S. Hence, there must exist a variable x_j, $j \in S' \setminus S$, with a negative reduced cost. □

The main advantage of the ICG algorithm is to benefit from the strengths of the ISUD algorithm for solving the RMP, namely, i) to exploit the RMP degeneracy in order to easily find integer descent directions and ii) to compute at almost every iteration an improved solution even if the CP is not solved at optimality at each iteration. This allows to compute a sequence of improving integer solutions and stop the algorithm whenever the quality of the current solution is satisfactory (assuming that a lower bound is previously computed to assess this quality).

In the ICG algorithm, column generation is applied to generate the variables of the SPP model (1)–(3). Standard CG uses the dual solution of the current RMP to define the subproblem objective function and generate negative reduced cost variables that can improve the current objective value, converging towards a lower bound. During CG, no effort is made to improve the current upper bound on the optimal value of the SPP. The proposed ICG algorithm is designed to do so. Indeed, instead of using the dual solution of the current RMP, it relies on a dual solution corresponding to the current integer solution which favors the generation of columns that can improve the current integer solution.

Finally, note that the columns accumulated while solving the subproblem with different dual solutions lead to an optimal solution even if these dual solutions are not necessarily of good quality. It is well known that poor-quality dual solutions have a negative impact on the convergence of standard CG algorithms. Given that the ICG algorithm can handle a very large number of columns in the RMP through the column pool and the multi-phase strategy, the quality of the dual solution has less impact on the algorithm convergence when a large number of columns is generated at each iteration. A possibility (not implemented) to further restrict the impact of the dual solution quality would be to solve in parallel several subproblems defined with different dual solutions.

4.2 An acceleration strategy

To speed up the solution process, we propose to generate whenever possible more than one integer descent direction each time that the CP is solved in Step 3 of Algorithm 1 when called from Steps 12 and 17 of Algorithm 2. Multiple directions can easily be handled in the RP if they are orthogonal. Therefore, after finding a first integer descent direction $\mathbf{d}_1$ defined according to (12), all variables λ_ℓ, $\ell \in S$, taking a positive value in the current solution of the CP is removed from the CP as well as all variables v_j, $j \in I_S$, such that $\lambda_\ell \, \mathbf{A}_\ell^\top \mathbf{A}_j \neq 0$ for at least one $\ell \in S$. In this way, any new solution to the CP gives a new direction that is orthogonal to $\mathbf{d}_1$. The updated CP is then solved. If it returns another integer descent direction $\mathbf{d}_2$, further variables are removed from the CP before solving it again, ensuring that only directions orthogonal to $\mathbf{d}_1$ and $\mathbf{d}_2$ can be generated. This process repeats until no integer descent direction can be found. When all directions are found, the current integer solution to the RP is updated before starting a new ISUD iteration. Generating multiple directions when solving a CP reduces the total number of ISUD iterations and, therefore, the total time to compute the degree of incompatibility of the generated columns and to build the CP at each iteration. Furthermore, given that variables are removed from the CP when each direction is found, the

updated CP can be solved more rapidly. Computational results reported in the next section will show the effectiveness of this strategy that we call the *Multi-Direction in the CP (MDCP) strategy*.

5 Computational results

In this section, we report computational results obtained by the ICG algorithm on VSCP and CPP instances. These two problems are modeled as SPPs that are subject to high degeneracy. The ICG algorithm is compared to two well-known dual-fractional heuristics (see, e.g., Joncour et al. [13]) that are based on CG, namely, a diving heuristic (DH) and a restricted master heuristic (RMH). DH is a branch-and-price heuristic that explores a single branch of the search tree. At each node where solving the MP results in a fractional-valued solution, it fixes to one the variable with the largest fractional value. The process stops when the solution of the MP is integer. RMH consists of solving by column generation the MP at the root node. Integrality requirements are then added on the variables contained in the last RMP. This integer RMP is then solved by a commercial MIP solver.

Based on preliminary experiments, we set the parameter values of the ICG Algorithm 2 as follows: $minImp = 0.0025$, $maxConsFail = 9$, and $maxDegree = 7$. Furthermore, the multi-phase strategy invoked in the ISUD Algorithm 1 executed the sequence of phases $k \in \{1, 2, \ldots, 5\}$, stopping thus before proving optimality.

All our tests were performed on a Linux machine equipped with an Intel Xeon E3-1226 processor clocked at 3.3GHz. All CPs, RMPs and mixed integer programs are solved by the IBM CPLEX commercial solver (version 12.4). For both VCSP and CPP, there are multiple column generation subproblems that are SPPRCs. They are solved by dynamic programming using the Boost library, version 1.54.

5.1 VCSP results

The VCSP is one of the important planning problems faced by transit companies. It consists of assigning simultaneously buses to bus trips and drivers to tasks which are defined by dividing each bus itinerary into segments linking consecutive possible driver relief points. In fact, for each bus trip, there is one task for each segment it contains, one task ensuring that a bus reaches the beginning of the trip and another ensuring that a bus leaves after the end of the trip. Traditionally, for large-sized instances, the problem has been tackled in two steps, assigning the buses first and the drivers afterwards. Several exact and heuristic algorithms have been developed to solve different variants (one or several depots, homogeneous or heterogeneous fleet, various duty types, etc.) of the integrated problem (see Ibarra-Rojas et al. [11]). We consider the single-depot, homogenous-fleet variant addressed by Haase et al. [10] who devised the first branch-and-price algorithm for the VCSP. We use the same model, except for the non-set-partitioning constraints that are omitted. There is one SPPRC subproblem for each of the two duty types. The networks underlying these subproblems are described in details in Haase et al. [10].

The random instance generator of Haase et al. [10] was used to generate the test instances. The size of an instance (number of tasks) is defined by the number of bus itineraries (B) and the number of relief points considered along each itinerary (R). It is equal to $B(R+1)$. For each pair (B, R), we generated three different instances using different seed numbers. Notice that, for the same instance number, the instances for a pair (B_1, R_1) and those for a pair (B_2, R_2) with $B_1 = B_2$ differ only by the number of relief points considered. Given that the number of reliefs in the middle of a bus trip tends to be minimized in an optimal solution, the optimal solutions of these instances are often the same. Nevertheless, the model contains a different number of constraints and the solution process varies. The set of instances is divided in two classes: small (800 tasks or less) and large (960 tasks or more).

For these instances, the initial solution $\mathbf{x}^0$ was defined in Step 1 of the ICG Algorithm 2 as an artificial one, where each task associated with a bus trip is covered by an artificial column bearing a large cost. In the initial dual solution $\boldsymbol{\alpha}^0$, each dual value associated with these tasks was then set to this large value divided by the number of tasks in the bus trip.

The computational results of some tests are reported in Tables 1 and 3 for the small and the large VCSP instances, respectively. For these first tests, the ICG algorithm did not use the MDCP strategy. In these tables, the first three columns describe the instance, namely, the number of tasks it contains (Tasks), the pair (B, R) defining it, and the instance number (No). Then, for each tested algorithm, they report the total computational time in seconds, the optimality gap in percentage between the cost of the best solution found and the linear relaxation optimal value, the number of column generation iterations (Itr), and the total number of columns generated (Col). For the RMH and ICG algorithms, the total number of integer solutions found (IntS) is also indicated. This number is not reported for DH because it is always equal to 1. Finally, the number of times that a mixed integer program was solved in Step 14 of Algorithm 2 (Mip) and the time (in seconds) spent solving these programs (MipT) are also specified for the ICG algorithm. Note that the linear relaxation optimal values are only computed for reporting the optimality gaps. The time required to compute them is not included in the total time. Note also that, for ICG, the number of columns generated does not include those that were added to the pool of columns but never considered in the RP or the CP because their degree of incompatibility remained too high.

Table 1: Results for the small VCSP instances

Instance			*RMH*					*DH*				*ICG*						
Tasks	**(B, R)**	***No.***	***Time***†	***Gap***†	***Itr.***	***Col.***	***IntS***	***Time***	***Gap***	***Itr.***	***Col.***	***Time***	***Gap***	***Itr.***	***Col.***	***IntS***	***Mip***	***MipT***
		1	0.9	0.0	12	2171	1	1.0	0.0	25	2714	0.7	0.0	6	9984	16	0	0.0
240	(40, 5)	2	0.6	0.0	16	1396	2	0.7	0.0	29	1582	0.3	0.0	5	7583	12	0	0.0
		3	1.0	0.0	10	1389	1	0.8	0.0	30	1886	0.6	0.0	6	8806	13	0	0.0
		1	1.2	0.0	11	2699	1	3.6	0.0	70	4766	1.1	0.0	6	9935	29	0	0.0
320	(80, 3)	2	1.1	0.0	12	2348	1	1.4	0.0	32	2658	0.7	0.0	7	8561	25	0	0.0
		3	1.4	0.0	14	2673	1	3.1	0.0	58	4284	1.2	0.0	7	10101	39	0	0.0
		1	6.8	0.0	12	4827	3	11.5	0.0	83	7804	4.5	0.0	6	35250	16	0	0.0
400	(40, 9)	2	5.8	0.0	16	3798	3	8.5	0.0	68	5912	2.8	0.0	5	24032	13	0	0.0
		3	30.8	0.0	14	3229	4	7.3	0.0	62	5022	3.8	0.0	5	27710	13	0	0.0
		1	5.9	0.0	15	4201	1	13.9	0.0	87	7545	4.5	0.0	7	21431	28	0	0.0
480	(80, 5)	2	5.2	0.0	16	3748	3	11.9	0.0	77	6421	2.6	0.0	6	17657	29	0	0.0
		3	6.9	0.0	18	3806	2	11.0	0.0	66	5527	6.1	0.0	8	64839	40	1	0.9
		1	16.5	0.0	17	6204	2	38.8	0.0	94	12036	14.5	0.0	6	33824	30	0	0.0
640	(80, 7)	2	15.3	0.0	20	5314	0	39.3	0.0	128	10255	9.7	0.0	6	37871	30	0	0.0
		3	38.5	0.0	20	6234	6	27.5	0.0	61	9147	10.9	0.0	6	43899	42	0	0.0
		1	3450.5	0.7	21	8377	5	58.5	0.0	114	15918	16.8	0.0	7	44955	63	0	0.0
720	(120, 5)	2	405.9	1.2	20	5675	4	39.8	2.8	102	10461	7.4	0.0	6	28790	40	0	0.0
		3	20.9	0.0	21	7674	1	45.7	0.0	114	12671	14.4	0.0	7	36479	62	0	0.0
		1	361.7	0.4	18	7470	5	168.2	0.0	214	25273	25.0	0.0	7	56143	29	0	0.0
800	(80, 9)	2	40.2	0.0	22	7972	2	131.0	0.0	166	20918	20.4	0.0	6	47682	30	1	1.2
		3	694.7	0.0	19	8079	4	152.9	0.0	174	25632	20.9	0.0	6	65417	40	0	0.0

†: All gaps are in percentage; all times are in seconds.

From Table 1, we observe that ICG finds the optimal solution for all instances in faster computational times than the other two heuristics. For the largest of these small instances, ICG can be as up to 7.3 times faster than DH and RMH. This is due to the small number of iterations performed by ICG (at most 7) compared to DH and the relatively large number of integer solutions found per iteration (on average, close to 5). RMH and DH also find an optimal solution for most instances. They fail to do so for 3 and 1 instances, respectively. It, thus, seems that omitting the non-set-partitioning constraints from the VCSP model of Haase et al. [10] makes the problem much easier to solve. Nevertheless, we observe that, for the largest of these small instances, the computational time required by RMH may be quite large (almost up to one hour). Finally, notice that, for these instances, ICG rarely needs to solve a mixed integer program to get a larger improvement of the objective value at a given iteration. This is also the case for the large VCSP instances as reported below.

One can remark that ICG uses much more columns than RMH and DH. To verify if this observation can explain the difference in the computational times, we ran additional tests with RMH and DH, increasing the number of columns generated at each iteration. This increase was achieved by not applying the dominance rule at the sink node of the networks and selecting a large number of the resulting columns based on their

reduced cost. The results of these tests are reported in Table 2. They show that, when a larger number of columns is generated (on average, the number of columns generated for DH is 1.2 times larger than the number used by ICG), RMH and DH become slower, and thus ICG remains the fastest algorithm. Notice that, in this case, an optimal solution can be found for all instances by both RMH and DH. In the following, the results for RMH and DH were obtained by generating a relatively small number of columns per iteration (as for the results in Table 1).

Table 2: RMH and DH results for small VCSP instances when generating many columns

Instance			*RMH*					*DH*			
Tasks	**(B, R)**	***No***	***Time***†	***Gap***†	***Itr***	***Col***	***IntS***	***Time***	***Gap***	***Itr***	***Col***
240	(40, 5)	1	2.6	0.0	9	4767	1	1.7	0.0	12	6131
		2	0.6	0.0	10	5141	6	0.6	0.0	10	5141
		3	1.2	0.0	6	3600	2	1.0	0.0	8	4664
320	(80, 3)	1	2.7	0.0	15	9000	1	2.8	0.0	15	9000
		2	1.6	0.0	10	5713	1	2.8	0.0	15	7828
		3	7.8	0.0	10	6000	2	4.7	0.0	14	8039
400	(40, 9)	1	242.7	0.0	15	9000	3	21.8	0.0	45	25210
		2	48.4	0.0	17	9901	2	12.2	0.0	39	21240
		3	6.4	0.0	15	8296	1	13.3	0.0	35	18391
480	(80, 5)	1	79.2	0.0	23	13470	1	30.8	0.0	51	28390
		2	35.1	0.0	17	10200	2	17.5	0.0	38	21916
		3	46.8	0.0	16	9600	2	27.8	0.0	46	26658
640	(80, 7)	1	4033.4	0.0	34	20400	3	143.2	0.0	135	77243
		2	2077.7	0.0	31	18517	3	70.1	0.0	77	43927
		3	4026.0	0.0	25	15000	5	82.1	0.0	77	42651
720	(120, 5)	1	54.5	0.0	40	23782	1	144.8	0.0	160	93565
		2	990.8	0.0	34	20228	3	129.0	0.0	99	57308
		3	4035.4	0.0	31	18600	3	123.0	0.0	111	60795
800	(80, 9)	1	3070.8	0.0	38	22800	1	363.3	0.0	159	90212
		2	4051.2	0.0	38	22800	3	380.4	0.0	111	64339
		3	4093.6	0.0	27	16200	2	249.3	0.0	109	64084

†: All gaps are in percentage; all times are in seconds.

For all the large VCSP instances, RMH was not able to find a feasible solution within a 1-hour time limit. Therefore, we report no results for RMH in Table 3. The results in this table confirm the superiority of ICG

Table 3: Results for the large VCSP instances

Instance			*DH*				*ICG*						
Tasks	**(B, R)**	***No.***	***Time***†	***Gap***†	***Itr.***	***Col.***	***Time***	***Gap***	***Itr.***	***Col.***	***IntS***	***Mip***	***MipT***
960	(160, 5)	1	150.43	0.0	158	23978	26.6	0.0	6	48437	92	0	0.0
		2	114.51	0.0	143	16778	24.9	0.0	7	50740	67	0	0.0
		3	110.00	0.0	123	18964	22.9	0.0	6	38647	75	0	0.0
1200	(200, 5)	1	347.39	0.0	220	34397	44.7	0.0	7	65955	103	0	0.0
		2	184.77	0.0	126	20985	52.1	0.0	8	190431	101	1	4.4
		3	242.82	0.0	183	27344	42.0	0.0	6	64578	105	0	0.0
1200	(120, 9)	1	1063.96	0.0	411	73736	101.4	0.0	6	121266	65	0	0.0
		2	748.59	0.0	334	48252	49.9	0.0	7	264412	40	1	8.5
		3	1015.50	0.0	354	53992	84.0	0.0	6	124171	58	0	0.0
1600	(160, 9)	1	2330.98	3.5	419	76463	170.6	0.0	6	140279	89	0	0.0
		2	1703.12	0.0	331	31955	150.6	0.0	7	120979	71	0	0.0
		3	2831.52	0.0	467	57719	144.2	0.0	7	117910	75	0	0.0
2000	(200, 9)	1	10081.60	0.0	968	296499	263.8	0.0	7	159891	107	0	0.0
		2	9345.04	0.0	873	155980	295.3	0.0	7	539911	100	1	20.5
		3	9169.62	0.0	990	192090	287.2	0.0	7	160804	112	1	18.3

†: All gaps are in percentage; all times are in seconds.

over DH. Indeed, one can observe that ICG is always faster and can yield remarkable time reductions of up to 97% for the largest tested instances (besides finding an optimal solution for all tested instances). The number of iterations performed by ICG (at most 8) is negligible compared to that achieved by DH (between 123 and 990), indicating that the large number of columns generated at each iteration is fully exploited by the ISUD algorithm when solving the RMP. This shows that strategies which aim at speeding up the ISUD algorithm and, in particular, the time required to solve the CP such as the MDCP strategy can be useful to increase the efficiency of ICG. To support this assertion, we conducted another series of experiments that consisted of solving the large VCSP instances with the ICG algorithm, but this time, applying the MDCP strategy. The results of these experiments are reported in Table 4. The last column in this table specifies the relative gain in computational time obtained by applying the MDCP strategy. We observe that the MDCP strategy yields substantial time reductions varying between 14.5% and 47.3%.

Table 4: Results of ICG with the MDCP strategy on the large VCSP instances

	Instance		*ICG*							
Tasks	**(B, R)**	***No.***	***Time***†	***Gap***†	***Itr.***	***Col.***	***IntS***	***Mip***	***MipT***	***Gain***†
960	(160, 5)	1	14.5	0.0	6	51040	92	0	0.0	45.5
		2	17.9	0.0	8	146408	49	1	3.0	28.1
		3	14.1	0.0	7	42520	74	0	0.0	38.4
1200	(200, 5)	1	38.2	0.0	8	77915	105	0	0.0	14.5
		2	31.8	0.0	7	143415	106	1	3.4	39.0
		3	26.4	0.0	6	62009	111	0	0.0	37.1
1200	(120, 9)	1	53.4	0.0	6	107706	66	0	0.0	47.3
		2	28.2	0.0	6	83243	41	0	0.0	43.5
		3	60.1	0.0	6	124428	60	0	0.0	28.5
1600	(160, 9)	1	99.7	0.0	6	160164	92	0	0.0	41.6
		2	92.8	0.0	7	123600	72	0	0.0	38.4
		3	95.7	0.0	7	134650	77	0	0.0	33.6
2000	(200, 9)	1	216.7	0.0	8	540545	110	1	24.8	17.9
		2	201.4	0.0	7	500187	100	1	20.3	31.8
		3	170.5	0.0	7	197081	117	0	0.0	40.6

†: All gaps and gains are in percentage; all times are in seconds.

5.2 CPP results

The CPP consists of finding least-cost crew pairings such that each flight of a given schedule is actively covered by a single pairing. A pairing is a sequence of flights performed by a crew that starts and ends at the crew base. For some of these flights, the crew may be deadheading, i.e., the crew members travel as passengers. To be feasible, a pairing must satisfy a variety of safety regulations and labor agreement rules. The CPP is usually solved by branch-and-price (see Desaulniers et al. [6]) where the pairings are generated by solving subproblems that can be modeled as SPPRCs. More details on the subproblems can be found in Saddoune et al. [20].

For our tests, we used five real-life CPP instances (without additional constraints) obtained from the datasets proposed by Kasirzadeh et al. [14]. Each instance is defined for a single aircraft fleet (D94, D95, 757, 319, or 320) and spans a single week. For the ICG Algorithm 2, an artificial solution was built as an initial solution $\mathbf{x}^0$ in Step 1: each task is covered by a single-task column that bears a very large cost. In the initial dual solution $\boldsymbol{\alpha}^0$, each dual value was set to this large cost.

Table 5 provides the computational results obtained on the CPP instances by the three heuristics, namely, RMH, DH, and ICG with the MDCP strategy. Its first two columns identify the instance by its name and the number of tasks it contains. Then, for each heuristic, it reports the same information as in Table 1.

Notice that, as for the VCSP instances, the number of column generation iterations executed by ICG is very small compared to DH. This can be explained by the large number of columns generated in each iteration. These columns are efficiently handled in the RMP through the multi-phase strategy used by the

ISUD algorithm. Furthermore, the multi-phase strategy applied at the subproblem level is useful to generate columns with a low incompatibility degree, yielding a low density coefficient matrix in the CP.

Table 5: Results for the CPP instances

Instance		*RMH*					*DH*				*ICG*						
No	*Tasks*	*Time*†	*Gap*†	*Itr*	*Col*	*IntS*	*Time*	*Gap*	*Itr*	*Col*	*Time*	*Gap*	*Itr*	*Col*	*IntS*	*Mip*	*MipT*
D94	424	15	0.05	22	17110	1	35	0.20	76	22217	17	0.14	12	32044	21	4	3
D95	1255	32849	1.13	56	100825	9	5411	0.21	914	313989	2164	0.22	20	341246	280	8	811
757	1290	41097	0.03	74	74221	4	18019	0.02	1203	597469	2175	0.01	15	396480	212	1	320
319	1293	60480	0.69	97	81335	8	2944	0.29	1268	306633	1218	0.18	23	283027	278	9	531
320	1740	30666	0.10	110	96188	5	4630	0.10	1269	367025	3806	0.05	20	328827	313	6	987

†: All gaps are in percentage; all times are in seconds.

One remarkable characteristic of the ICG algorithm is the large number of different integer solutions found throughout the solution process (more than 10 per iteration for the largest instances). This feature is highly desirable in the industry because it allows to stop the solution process at any time after finding a first satisfactory solution. To determine the quality of a solution, one may compute a lower bound in parallel. For the tested instances, this requires less than 50% of the time required by the ICG algorithm.

Finally, we observe that the number of times that a mixed integer program was solved in Step 14 of Algorithm 2 is small and the time devoted to solving these programs varied between 15% and 44% of the total time. Despite the large proportion of time that it may consume, this step has proven to be useful to accelerate the solution process.

Figure 2 depicts the cost of the current solution in function of the computational time for instance 320. Each point corresponds to a solution found during the solution process. The blue circles represent those obtained by the ISUD algorithm while the red plus signs represent those produced by solving a mixed integer program. We can observe a rapid cost decrease at the beginning of the solution process. Like traditional column generation methods, the cost decrease becomes slow towards the end. Notice that the solutions obtained by solving a mixed integer program (especially the first one) can sometimes yield a large cost decrease. We would like to point out that, for all instances, the algorithm succeeded to find a first feasible integer solution within the first few iterations of the solution process.

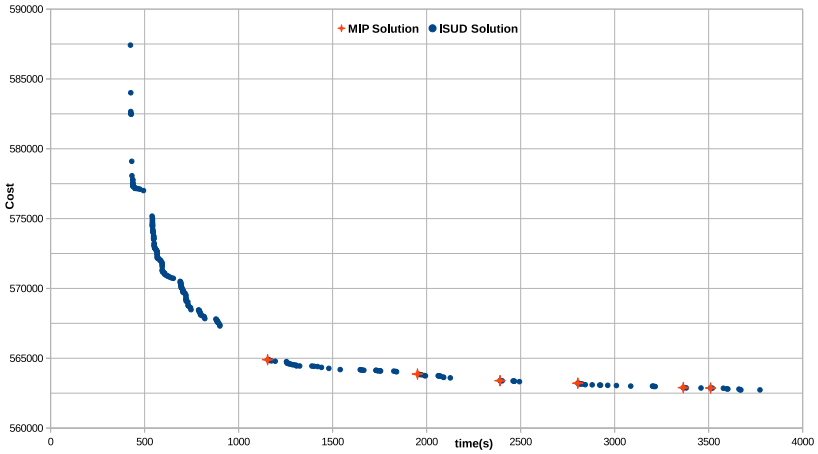

Figure 2: Current solution cost in function of the time (instance 320)

To conclude this section, we report in Figure 3 the number of pairings selected in the final solution that have already been generated at each iteration of the solution process for both DH and ICG. We observe that much more selected pairings are generated per iteration with ICG. This is not surprising given that ICG generates much more columns per iteration than DH, but also that the dual solutions used to generate columns in ICG correspond to the current integer solution and increase the chances of generating columns that can be part of better solutions.

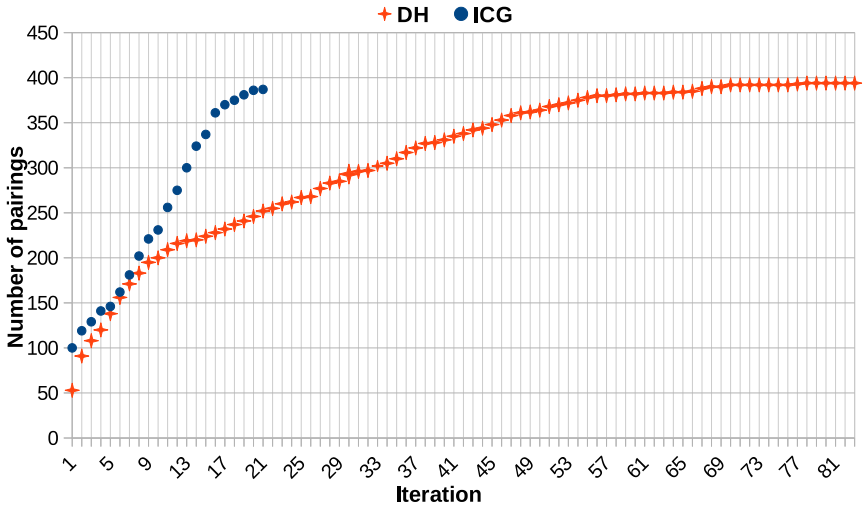

Figure 3: Number of pairings from the final solution already generated at each iteration (instance 320)

6 Conclusion

In this paper, we introduced the ICG algorithm which combines column generation and the recent ISUD algorithm for solving SPPs involving a very large number of variables. This primal algorithm generates a sequence of integer solutions with decreasing costs until reaching an optimal or near-optimal solution. It benefits from generating a large number of columns at each column generation iteration and exploits at each iteration a dual solution that corresponds to the current integer solution and favors the generation of columns that can be part of improved integer solutions. Our computational experiments on VCSP and CPP instances involving up to 2000 set partitioning constraints showed that ICG outperforms two popular column generation heuristics, producing for almost all instances better quality solutions in less computational times. For the largest VCSP instances, ICG can yield time reductions as large as 97%.

Several research avenues can be explored in the future. One of them consists of speeding up the proposed ICG algorithm by considering better columns in the CP, either by influencing the generation of the columns from the subproblems or by better selecting them from the pool of columns. Machine learning tools may be helpful to achieve this goal. Another important research direction is to generalize the ICG algorithm for solving SPPs with additional constraints.

References

[1] Balas, E., Padberg, M. W. (1972). On the set-covering problem. Operations Research, 20(6), 1152–1161.

[2] Balas, E., Padberg, M. (1975). On the set-covering problem: II. An algorithm for set partitioning. Operations Research, 23(1), 74–90.

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

[4] Bouarab H., Elhallaoui, I., Metrane, A., Soumis, F. (2017). Dynamic constraint and variable aggregation in column generation. European Journal of Operational Research 262(3), 835-850.

[5] Dantzig, G. B., Wolfe, P. (1960). Decomposition principle for linear programs. Operations Research, 8(1), 101–111.

[6] Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M. M., Soumis, F. (1997). Crew pairing at Air France. European Journal of Operational Research, 97(2), 245–259.

[7] Elhallaoui, I., Metrane, A., Desaulniers, G., Soumis, F. (2011). An improved primal simplex algorithm for degenerate linear programs. INFORMS Journal on Computing, 23(4), 569–577.

[8] Elhallaoui, I., Villeneuve, D., Soumis, F., Desaulniers, G. (2005). Dynamic aggregation of set-partitioning constraints in column generation. Operations Research, 53(4), 632–645.

[9] Elhallaoui, I., Metrane, A., Soumis, F., Desaulniers, G. (2010). Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming, 123(2), 345–370.

[10] Haase, K., Desaulniers, G., Desrosiers, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. Transportation Science, 35(3), 286–303.

[11] Ibarra-Rojas, O., Delgado, F., Giesen, R., Muñoz, J. (2015). Planning, operation, and control of bus transport systems: A literature review. Transportation Research Part B, 77(1), 38–75.

[12] Irnich, S., Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers, G., Desrosiers, J., Solomon, M.M., (eds.), *Column Generation*, chapter 2, pages 33–65. Springer, New York.

[13] Joncour, C., Michel, S., Sadykov, R., Vanderbeck, F. (2010). Column generation based primal heuristics. Electronic Notes in Discrete Mathematics, 36, 695–702.

[14] Kasirzadeh, A., Saddoune, M., Soumis, F. (2017). Airline crew scheduling: Models, algorithms, and data sets. EURO Journal on Transportation and Logistics, 6(2), 111–137.

[15] Lübbecke, M. E., Desrosiers, J. (2005). Selected topics in column generation. Operations Research, 53(6), 1007–1023.

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

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

[18] Rosat, S., Elhallaoui, I., Soumis, F., Chakour, D. (2016). Influence of the normalization constraint on the integral simplex using decomposition. Discrete Applied Mathematics, 217(1), 53–70.

[19] Rosat, S., Elhallaoui, I., Soumis, F., Lodi, A. (2017). Integral simplex using decomposition with primal cutting planes. Mathematical Programming, doi:10.1007/s10107-017-1123-x.

[20] Saddoune, M., Desaulniers, G., Soumis, F. (2013). Aircrew pairings with possible repetitions of the same flight number. Computers & Operations Research, 40(3), 805–814.

[21] Saxena, A. (2003) Set-partitioning via integral simplex method, Carnegie Mellon University, USA.

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

[23] Trubin, V. A. (1969). On a method of solution of integer linear programming problems of a special kind. Soviet Mathematics Doklady, 10, 1544–1546.

[24] Vanderbeck, F. (2005). Implementing mixed integer column generation. In Desaulniers, G., Desrosiers, J., Solomon, M.M., (eds.), *Column Generation*, chapter 12, pages 331–358. Springer, New York.

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

[26] Zaghrouti, A., Soumis, F., El Hallaoui, I. (2013). Improving ILP Solutions by Zooming Around an Improving Direction. Cahiers du Gerad, G-2013-107, HEC Montréal.