Operations Research

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

Integral Simplex Using Decomposition for the Set Partitioning Problem

Abdelouahab Zaghrouti, François Soumis, Issmail El Hallaoui

To cite this article:
Abdelouahab Zaghrouti, François Soumis, Issmail El Hallaoui (2014) Integral Simplex Using Decomposition for the Set Partitioning Problem. Operations Research 62(2):435-449. https://doi.org/10.1287/opre.2013.1247

OPERATIONS RESEARCH
Vol. 62, No. 2, March–April 2014, pp. 435–449
ISSN 0030-364X (print) | ISSN 1526-5463 (online)

http://dx.doi.org/10.1287/opre.2013.1247

Integral Simplex Using Decomposition for the Set Partitioning Problem

Abdelouahab Zaghrouti, François Soumis, Issmail El Hallaoui
Département de Mathématiques et Génie Industriel, GERAD, and Polytechnique Montréal, Montréal, Quebec H3C 3A7, Canada
{abdelouahab.zaghrouti@gerad.ca, francois.soumis@gerad.ca, issmail.elhallaoui@gerad.ca}

Since the 1970s, several authors have studied the structure of the set partitioning polytope and proposed adaptations of the simplex algorithm that find an optimal solution via a sequence of basic integer solutions. Balas and Padberg in 1972 proved the existence of such a sequence with nonincreasing costs, but degeneracy makes it difficult to find the terms of the sequence. This paper uses ideas from the improved primal simplex to deal efficiently with degeneracy and find subsequent terms in the sequence. When there is no entering variable that leads to a better integer solution, the algorithm referred to as the integral simplex using decomposition algorithm uses a subproblem to find a group of variables to enter into the basis in order to obtain such a solution. We improve the Balas and Padberg results by introducing a constructive method that finds this sequence by only using normal pivots on positive coefficients. We present results for large-scale problems (with up to 500,000 variables) for which optimal integer solutions are often obtained without any branching.

Subject classifications: integral simplex; set partitioning problem; decomposition.
Area of review: Optimization.
History: Received October 2011; revisions received March 2013, October 2013; accepted October 2013. Published online in *Articles in Advance* February 20, 2014.

1. Introduction

Consider the set partitioning problem (SPP)

$$\begin{aligned} &\min \quad cx \\ (\mathbb{P}) \quad &Ax = e \\ &x_j \text{ binary}, \quad j \in \{1, \ldots, n\}, \end{aligned}$$

where A is an $m \cdot n$ binary matrix, c an arbitrary n-vector, and $e = (1, \ldots, 1)$ an m-vector. Let $(\mathbb{P}')$ be the linear program obtained from $(\mathbb{P})$ by replacing x_j binary with $x_j \geqslant 0$, $j \in \{1, \ldots, n\}$. We can assume without loss of generality that A is full rank and has no zero rows or columns and does not contain identical columns. We denote by A_j the jth column of A. Also, for a given basis B we denote $\bar{A}_j = (\bar{a}_{ij})_{1 \leqslant i \leqslant m} = B^{-1}A_j$.

This article uses some results of Balas and Padberg (1972), which are presented in detail in §2.1. We first use their results on the existence of a sequence of basic integer solutions, connected by simplex pivots, from any initial integer solution to a given optimal integer solution. The costs of the solutions in this sequence form a descending *staircase*. The vertical section of a step corresponds to a basic change leading to a decrease in the cost of the solution. Each step generally contains several basic solutions associated with the same extreme point of the polyhedron of $\mathbb{P}'$. These solutions are obtained by a sequence of degenerate pivots forming the horizontal section of a step.

We also use the results of Balas and Padberg (1975) on the necessary and sufficient conditions for a set of columns to ensure a transition from one integer solution to another one when they are entered into the basis. A minimal set satisfying these conditions permits to move to an adjacent integer extreme point. The authors also present an enumeration method to find minimal sets that allow the transition from one integer solution to a better adjacent integer solution.

Several other authors have presented enumeration methods that move from one integer solution to a better adjacent integer solution: Haus et al. (2001), Thompson (2002), Saxena (2003), and Rönnberg and Larsson (2009). However, these enumeration methods, moving from a basis to an adjacent (often degenerate) basis, need in practice computation times growing exponentially with the size of the problem, mainly due to severe degeneracy.

This article also uses results on the improved primal simplex (IPS) algorithm for degenerate linear programs (Elhallaoui et al. 2011, Raymond et al. 2010). These results are presented in more detail in §2.2. This algorithm separates the problem into a reduced problem (*RP*) containing nondegenerate constraints and a complementary problem (*CP*) containing degenerate constraints. The latter generates combinations of variables that improve the reduced problem solution. These variable combinations are minimal because they do not contain strict subsets that permit the improvement of the solution.

Section 3 starts with a global view of the integral simplex using decomposition algorithm (ISUD) we propose in this paper and contains theoretical results proving its validity. Each subsection explains and justifies a procedure of the

algorithm. We discuss the relationships between the combinations of variables generated by the complementary problem of IPS and the minimal sets of Balas and Padberg (1975). We present the conditions to be added to the complementary problems to obtain combinations of columns that permit us to move from one integer solution to a better one. Interestingly, the generated combinations often satisfy the conditions and are generally small. When the conditions are not satisfied, we can use a branching method for the complementary problem to satisfy them. This leads to an efficient method for obtaining combinations of columns that permit moving from an integer solution to a better one.

Section 4 presents improvements to Balas and Padberg's (1972, 1975) theoretical results on the sequence of adjacent integer bases permitting to move from an initial integer solution to an optimal integer solution. We show that ISUD uses ordinary pivots on positive coefficients only.

Section 5 presents ideas permitting to obtain an efficient first implementation of the algorithm. This section contains empirical observations on which some strategies are based to speed up the algorithm.

Section 6 presents numerical results for bus driver and aircrew scheduling problems with up to 500,000 variables and 1,600 constraints. We show that ISUD is able to find (but not necessarily prove) optimal solutions to the hardest instances in less than 20 minutes. CPLEX was not able to find any feasible solution to such difficult instances in 10 hours.

2. Preliminaries

2.1. Results from Balas and Padberg

Balas and Padberg (1972) established the following results. Two bases for a linear program are called adjacent if they differ in exactly one column. Two basic feasible solutions are called adjacent if they are adjacent vertices of the convex polytope that is the feasible set. This distinction is necessary, since two adjacent bases may be associated with the same solution, and two adjacent (basic) feasible solutions may be associated with two nonadjacent bases.

The solution associated with a feasible basis B is integer if and only if there exists Q, a subset of columns of B, such that $\sum_{j\in Q} A_j = e$. If A is of full row rank, every feasible integer solution to $\mathbb{P}'$ is basic.

For $i = 1, 2$, let x^i be a basic feasible integer solution to $\mathbb{P}'$, B_i an associated basis, I_i and J_i the basic and nonbasic index sets, and $Q_i = \{j \mid x_j = 1\}$.

THEOREM 3.1 (*Balas and Padberg* 1972). *If x^2 is an optimal solution to $\mathbb{P}$, then there exists a sequence of adjacent bases $B_{10}, B_{11}, B_{12}, \ldots, B_{1p}$ such that $B_{10} = B_1$, $B_{1p} = B_2$, and*

(a) *the associated basic solutions $x^1 = x^{10}, x^{11}, x^{12}, \ldots, x^{1p} = x^2$ are all feasible and integer,*

(b) $c \cdot x^{10} \geqslant \cdots \geqslant c \cdot x^{1p}$, *and*

(c) $p = |J_1 \cap Q_2|$.

In the proof of this theorem we see that the column pivoted in may have null reduced cost ($\bar{c}_j = 0$) and pivots on negative $\bar{a}_{ij}$ may appear.

Balas and Padberg (1975) observed the difficulty of identifying the improving sequence of pivots when the optimal solution x^2 is unknown. The authors stated, "Since set partitioning problems tend to be highly degenerate, the feasible polytope usually contains an enormous number of vertices adjacent to a given vertex. Furthermore, lexicographic or similar techniques are of no avail in coping with degeneracy, since the sequence of pivots required to reach an adjacent vertex may include pivots on a negative entry in a degenerate row" (Balas and Padberg 1975, p. 75).

The article establishes relationships between two adjacent or nonadjacent integer solutions, and the sets of columns on which we must carry out pivots in order to move from one to the other.

THEOREMS 1 AND 2 (*Balas and Padberg* 1975) (*without the Tucker tableau notation*). *Let x^1 be a basic integer solution and I_1, J_1, and Q_1 the associated index sets.*

There exists $Q \subseteq J_1$ such that

$$Q^+ = \left\{ k \;\middle|\; \sum_{j\in Q} \bar{a}_{kj} = 1 \right\} \subseteq Q_1, \tag{1}$$

$$Q^- = \left\{ k \;\middle|\; \sum_{j\in Q} \bar{a}_{kj} = -1 \right\} \subseteq I_1 \cap \bar{Q}_1 \tag{2}$$

$$\sum_{j\in Q} \bar{a}_{kj} = 0 \subseteq I_1 \backslash \{Q^+ \cup Q^-\} \tag{3}$$

if and only if

$$x_j^2 = \begin{cases} 1 & \text{if } j \in Q \cup Q^- \cup (Q_1 - Q^+) \\ 0 & \text{otherwise} \end{cases}$$

is a feasible integer solution obtained from x^1 by performing pivots on Q.

REMARK 1. The columns associated with (entering) variables in $Q \cup Q^-$ are disjoint, i.e., not covering the same set partitioning constraints and these entering variables replace those in Q^+ (the leaving variables).

A set $Q \subseteq J_1$ for which (1)–(2) hold will be called decomposable if Q can be partitioned into subsets Q^* and Q^{**} such that (1)–(3) hold for both Q^* and Q^{**}. That means that columns with indexes in $Q \cup Q^-$ will replace those with indexes in Q^+ in x^1 in order to obtain an improved solution x^2.

THEOREM 3 (*Balas and Padberg* 1975). *Let x^1 and x^2 be two integer solutions to $\mathbb{P}$, with $Q = J_1 \cap Q_2$. Then x^2 is adjacent to x^1 if and only if Q is not decomposable.*

The paper (Balas and Padberg 1975) describes procedures for generating all-integer vertices of x adjacent to a

given vertex. Unfortunately, the procedures are combinatorial and can not solve large problems.

Corollaries 3.2, 3.3, *and* 3.5 (*Balas and Padberg* 1975). Let x^1 and x^2 be two nonadjacent vertices in the feasible domain of $\mathbb{P}$ related to each other by pivots on $Q = J_1 \cap Q_2$.

(a) Then there exists a partition of Q, $Q = \bigcup_{i=1}^{h} Q_i$, such that Q_i satisfies (1)–(2) and Q_i is not decomposable.

(b) For any $H \subseteq \{1, \ldots, h\}$ the pivots on $\bigcup_{i \in H} Q_i$ starting from x^1 reach a feasible integer solution to $\mathbb{P}$.

(c) Any permutation of $\{1, \ldots, h\}$ defines a path of adjacent vertices from x^1 to x^2.

2.2. Results from the Improved Primal Simplex

We present the results of Elhallaoui et al. (2011) using the notation introduced by Balas and Padberg (1972, 1975), and Raymond et al. (2010). We discuss these results for $\mathbb{P}'$, the linear relaxation of the set partitioning problem, although they were developed for a general linear programming problem.

The following notation is used (*please note the I and J used in this section for illustrating IPS are different from the I and J used by Balas and Padberg 1972, 1975*). If $x \in R^n$ and $I \subseteq \{1, \ldots, n\}$ is an index set, x_I denotes the subvector of x indexed by I. Similarly for the $m \cdot n$ matrix A, A_I denotes the $m \cdot |I|$ matrix whose columns are indexed by I. If $J = \{1, \ldots, n\} \backslash I$, $x = (x_I, x_J)$ is used even though the indices in I and J may not appear in order. In the same way the superior indices denote the subset of rows. The vector of all ones with dimension dictated by the context is denoted e.

The paper defines a reduced problem with nondegenerate constraints. This problem is defined according to a feasible basis B that provides the basic feasible solution $\bar{x} = B^{-1}b$. Let P be the index set of the p positive components of this solution. Most of the time, a basic solution to a set partitioning problem is degenerate and $p < m$. Let N be the index set of the zero basic variables. With the partition (P, N) of the constraints and after multiplying by B^{-1} the linear system $Ax = b$ becomes

$$\begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} \begin{bmatrix} x^P \\ x^N \end{bmatrix} = \begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix}, \tag{4}$$

with $\bar{b}^N = 0$, $\bar{b}^P > 0$, and $\bar{A}^N_j = 0$, $\forall\, j \in P$.

Definition 1. The jth variable x_j of $\mathbb{P}'$ is said to be compatible with this basis if and only if $\bar{A}^N_j = 0$. Otherwise, it is said to be incompatible.

It is worthy to outline, as observed in Metrane et al. (2010), that a variable is equivalently said to be compatible with a basis if it is linearly dependent on the positive (nondegenrate) variables (columns) of this basis. Please note that the terms "variable" and its associated "column" are used interchangeably throughout the paper. It is this observation that is used to prove Proposition 6. Let $C \subseteq \{1, \ldots, n\}$ be the index set of compatible variables and $I = \{1, \ldots, n\} \backslash C$ the index set of incompatible variables; x_C and x_I are the subvectors of x of compatible and incompatible variables, respectively. The cost vector is partitioned into c_C and c_I and the columns of A into A_C and A_I.

The linear system (4) can be rewritten as follows:

$$\bar{A}^P_C x_C + \bar{A}^P_I x_I = \bar{b}^P$$
$$\bar{A}^N_C x_C + \bar{A}^N_I x_I = 0.$$

The reduced problem RP is obtained by imposing $x_I = 0$. Hence,

$$\begin{aligned} &\min_{x_c} \quad c_C \cdot x_C \\ &\text{subject to} \quad \bar{A}^P_C x_C = \bar{b}^P, \quad x_C \geqslant 0 \end{aligned} \qquad (RP).$$

When $x_I = 0$ the constraints indexed by N are satisfied because $\bar{A}^N_C = 0$.

Observations.

- A compatible variable with negative reduced cost enters into the basis with a nondegenerate pivot and a lower cost solution is obtained.
- An optimal solution x^*_C of RP can easily be obtained with the primal simplex. Each pivot moves from an extreme point to an adjacent extreme point with a lower cost. The set P is reduced if degenerate constraints appear.
- Compared to a previous feasible solution, $x = (x^*_C, 0)$ is an improved feasible solution to $\mathbb{P}'$.

In §3 we will discuss the optimization of RP when the starting solution $\bar{x}$ is integer. The complementary problem is defined when RP is optimal and P, N, C, and I have been readjusted if necessary. The complementary problem is defined with the degenerate constraints, the incompatible variables, and the reduced cost vector $\bar{c}_I$ in the following way:

$$\begin{aligned} Z^{CP} = \min_{v} \quad & \bar{c}_I \cdot v \\ \text{subject to} \quad & \bar{A}^N_I v = 0, \quad e \cdot v = 1, \quad v \geqslant 0 \end{aligned} \qquad (CP)$$

where $\bar{c}_I = c_I - c_P \bar{A}^P_I$.

The one-norm bounding constraint ($e \cdot v = 1$) also called convexity constraint plays an important role in the efficiency of the proposed algorithm, as will be discussed in §5. It helps finding small combinations of columns that are often disjoint.

Proposition 2 (*Elhallaoui et al.* 2011). *If CP is infeasible or $Z^{CP} \geqslant 0$ then $(x^*_C, 0)$ is an optimal solution to $\mathbb{P}'$.*

Lemma 1 (*Elhallaoui et al.* 2011). *If CP is feasible, v^* is an optimal solution with $Z^{CP} < 0$, and S is the subset of indices for which $v^*_j > 0$, then adding the variables in S to RP reduces the cost of the solution.*

Proposition 3 (*Elhallaoui et al.* 2011). *The convex combination $w = \sum_{j \in S} A_j v^*_j$ is compatible with RP and the set S is minimal in the sense that no convex combination of a strict subset of S is compatible with RP.*

Elhallaoui et al. (2010) presented an efficient algorithm for the linear relaxation of the set partitioning problem using compatible variables and a reduced problem *RP*. This paper contains an interesting result for us.

PROPOSITION 5.6 (*Elhallaoui et al.* 2010). *If x^1 is a nondegenerate feasible integer solution to RP and if there exists a compatible variable with a negative reduced cost, then the new solution obtained by pivoting on this variable is an improved integer solution.*

3. The Integral Simplex Using Decomposition Algorithm

Balas and Padberg (1972) showed that between two integer solutions there exists a sequence of integer solutions with nonincreasing costs for which the associated bases are adjacent. However, some pivots between adjacent bases must be carried out on negative coefficients. We propose the integral simplex using decomposition, a new algorithm based on the IPS decomposition to find a sequence of adjacent basic solutions of nonincreasing costs leading to optimality. Furthermore, the algorithm carries out simplex pivots only on the positive coefficients. This point is discussed in §4.

ISUD Algorithm (Global View)

Step 0. Start with an initial integer solution.

Step 1.

- Improve the current integer solution with efficient pivots in the reduced problem (see §3.1 for details).
- Each pivot produces a better integer solution by increasing one variable. This creates a step in the staircase with a single basis change to move to a better adjacent extreme point.
- When there is no improvement with Step 1, go to Step 2.

Step 2.

- Improve the current integer solution with a solution to the complementary problem: a group of disjoint variables having (together) a negative reduced cost (see §3.2 for details).
- This step produces a better integer solution by increasing many variables. This creates a step in the staircase with many basis changes by degenerate pivots before a nondegenerate pivot, permitting to move to a better adjacent extreme point, is executed.
- Some branching can be necessary in this step to obtain a column-disjoint solution to *CP* (see §3.3 for details).

Control step:

- If Step 2 improves the solution, go to Step 1.
- If there is no more improvement with Step 2, the integer solution is optimal (see §3.4 for details).

We first add a result to the work of Balas and Padberg (1972). The result is an extension of a basic result of linear programming on the transition between two adjacent basic solutions. When x^2 is obtained from x^1 by a simplex pivot entering x_j into the basis at value 1, we know that

$$c \cdot x^2 = c \cdot x^1 + \bar{c}_j = c \cdot x^1 + c_j - c_B \bar{A}_j,$$

where c_B is the cost of the basic variables and $\bar{c}_j$ is the reduced cost of variable j. We extend this result to a move from x^1 to x^2 with a sequence of pivots on variables of a set Q. Generally, some reduced costs are modified after each pivot and the cumulative effect can be complex. In our case, the properties of the set Q as explained below permit to prove the following result.

PROPOSITION 4. *If x^1 and x^2 are two integer solutions related by a sequence of pivots on the variables of a set Q satisfying* (1)–(3)*, then*

$$c \cdot x^2 = c \cdot x^1 + \sum_{j \in Q} c_j - c_B \cdot \sum_{j \in Q} \bar{A}_j.$$

PROOF. Consider the aggregated variable that is the sum of the (entering) variables in $Q \cup Q^-$. Its reduced cost is $\sum_{j \in Q \cup Q^-} \bar{c}_j$. As outlined in Remark 1, pivoting on variables in Q has the same effect as pivoting on the aggregated variable. So, we have

$$c \cdot x^2 = c \cdot x^1 + \sum_{j \in Q \cup Q^-} \bar{c}_j.$$

As the reduced cost of variables in Q^- is null because they are basic, we obtain the desired result:

$$c \cdot x^2 = c \cdot x^1 + \sum_{j \in Q} c_j - c_B \cdot \sum_{j \in Q} \bar{A}_j. \quad \square$$

3.1. Improvement of the Current Integer Solution by *RP*

This subsection explains and justifies Step 1 of the algorithm. The IPS method is here specialized for set partitioning problems starting with an initial integer solution. In this case, we will see that the optimization of the reduced problem involves carrying out pivots on variables that move from one integer solution to a better one.

Let x be an integer solution to the set partitioning problem and P the index set of its positive components. Define an associated basis for which the first p variables are those of P and the remaining $m - p$ are artificial variables of cost M; M is sufficiently large to ensure that the artificial variables do not appear in an optimal solution. The constraints are also partitioned into (P, N), where P is the index set of the nondegenerate constraints and N the index set of the degenerate constraints. The same set P is used for variables and constraints because it is the same set after reordering the constraints and variables to obtain $A_P^P = I$. The basis with this partition is

$$B = \begin{bmatrix} A_P^P & A_N^P \\ A_P^N & A_N^N \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_P^N & I \end{bmatrix}. \quad B^{-1} = \begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \quad \text{because}$$

$$\begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \begin{bmatrix} I & 0 \\ A_P^N & I \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_P^N - A_P^N & I \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}.$$

Multiplying by B^{-1} the system $Ax = b$ becomes

$$\begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} \begin{bmatrix} x^P \\ x^N \end{bmatrix} = \begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix},$$

where

$$\begin{bmatrix} \bar{A}^P_j \\ \bar{A}^N_j \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A^N_P & I \end{bmatrix} \begin{bmatrix} A^P_j \\ A^N_j \end{bmatrix} = \begin{bmatrix} A^P_j \\ -A^N_P \quad A^P_j + A^N_j \end{bmatrix}, \tag{5}$$

$\bar{A}^N_j = 0 \ \forall j \in P$ because $A^P_P = I$, and

$$\begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A^N_P & I \end{bmatrix} \begin{bmatrix} b^P \\ b^N \end{bmatrix} = \begin{bmatrix} b^P \\ -A^N_P \quad b^P + b^N \end{bmatrix} = \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

By definition, the jth variable x_j of $\mathbb{P}$ is said to be compatible with the basis B if and only if $\bar{A}^N_j = 0$.

Using C and I, the index sets of compatible and incompatible variables, x_C and x_I are a partition of the variables, c_C and c_I are a partition of the variable costs, and A_C and A_I are a partition of the columns of A. The reduced problem RP is obtained by imposing $x_I = 0$. It is

$$\begin{aligned} \min_{x_C} \ & c_C \cdot x_C \\ \text{s.t.} \ & \bar{A}^P_C x_C = \bar{b}^P, \quad x_C \geqslant 0. \end{aligned} \tag{RP}$$

In this case the basis $A^P_P = I$, $\bar{b}^P = e$, and $x_P = Ix_P = e$.

Proposition 2. *If x is a nonoptimal integer solution to RP then there exists a compatible variable with negative reduced cost and the new solution obtained by pivoting on this variable is an improved integer solution.*

Proof. If x is nonoptimal then there exists a compatible variable with negative reduced cost. Furthermore, if x is integer, feasible, and nondegenerate then by Proposition 5.6 (Elhallaoui et al. 2010) the new solution obtained by pivoting on this variable is an improved integer solution. □

Proposition 3. *If we start with a feasible integer solution to RP, x^1, and apply the simplex algorithm while eliminating the degenerate constraints and the resulting incompatible columns as they appear, we generate a sequence $x^1, x^2, \ldots, x^k$ of solutions with the following properties*:

1. $c \cdot x^1 > c \cdot x^2 > \cdots > c \cdot x^k$;
2. *$x^1, x^2, \ldots, x^k$ are integer solutions of $\mathbb{P}$;*
3. *x^k is an optimal solution to the linear relaxation of the final RP problem.*

Proof. Given x^i, let RP^i be the associated reduced problem. If there exists a compatible variable with a negative reduced cost, the solution x^{i+1} after a simplex pivot is integer and $cx^i > cx^{i+1}$ by Proposition 5. RP^{i+1} is defined by eliminating the degenerate constraints if they exist. The process continues while there is at least one compatible variable with a negative reduced cost.

The solutions x^i of the RP^i problems are solutions of $\mathbb{P}$ if we complete them by adding incompatible variables set to zero values.

x^k is an optimal solution to the linear relaxation of the problem RP^k because there are no more compatible variables with negative reduced costs. □

However, this optimal solution to RP might not be an optimal solution to $\mathbb{P}$. In the following subsection, we discuss how to improve it.

3.2. Improvement of the Current Integer Solution by *CP*

This section explains and justifies Step 2 of the algorithm in the basic case when the branching is not necessary. Section 3.3 discusses the branching procedure called when necessary.

We now suppose that x is an optimal integer solution to RP and RP has been reduced as necessary so that x is nondegenerate, and B is the associated basis after completing P with artificial variables. The optimization of the complementary problem is used to find variables that permit the improvement of the integer solution if they are all entered into the current basis. The complementary problem CP is

$$\begin{aligned} Z^{CP} = \min \ & \bar{c}_I \cdot x_I \\ \text{s.t.} \ & \bar{A}^N_I x_I = 0, \quad e \cdot x_I = 1, \quad x_I \geqslant 0. \end{aligned} \tag{CP}$$

In this case,

$$\bar{A}^N_I = -A^N_P A^P_I + A^N_I,$$
$$\bar{c}_I = c_I - (A^P_P)^{-1} A^P_I c_P = c_I - A^P_I c_P.$$

Proposition 4. *If x^*_C is an optimal integer solution to RP and CP is infeasible or $Z^{CP} \geqslant 0$ then $(x^*_C, 0)$ is an optimal solution to $\mathbb{P}$.*

Proof. $(x^*_C, 0)$ is an optimal solution to $\mathbb{P}'$ by Proposition 5 (Elhallaoui et al. 2011). Since $\mathbb{P}'$ is a relaxation of $\mathbb{P}$, this integer solution is an optimal solution to $\mathbb{P}$. □

Let x^*_I be an optimal solution to CP and S the set of indices $j \in I$ such that $x^*_j > 0$.

Proposition 5. *The convex combination $w = \sum_{j \in S} A_j x^*_I$ is compatible with RP, and the set S is minimal in the sense that no convex combination of a strict subset of S is compatible with RP.*

Proof. This follows from Proposition 6 (Elhallaoui et al. 2011). □

We introduce the following definition before establishing the relationship between the sets S of IPS and the sets Q of Balas and Padberg (1972, 1975).

Definition 2. Let $S \subseteq I$ be a set of columns, then S is column disjoint if $A_{j_1} \cdot A_{j_2} = 0, \ \forall (j_1, j_2) \in S \times S, \ j_1 \neq j_2$.

For Propositions 6, 7, 10, and 11, consider the following context: Let x^*_C be an optimal solution to RP having a set of constraints P, x^*_I an optimal solution to CP, and S the set of indices $j \in I$ such that $x^*_j > 0$. Consider the case where $\bar{c}_I \cdot x^*_I < 0$ and S is column disjoint.

PROPOSITION 6. *The value the nonzero variables x_j^*, $j \in S$ take in an optimal solution to CP is $x_j^* = 1/|S|$ for $j \in S$.*

PROOF. Every solution to *CP* is compatible. So, we can say that this solution is a linear combination of the positive variables of the basis, i.e., the columns of S. Mathematically speaking, $\sum_{j\in S} x_j A_j = \sum_{k\in P} \lambda_k A_k$, where $(\lambda_k)_{k\in P}$ are reals. The normalization constraint must be satisfied, meaning that $\sum_{j\in S} x_j = 1$. The optimal solution to *CP* we are looking for is integer, i.e., its columns are disjoint. We also know that the current solution to *RP* (and also to the original problem) is integer. So, we can say that the columns of this solution are disjoint too. From all of that, we easily can see that $x_j^* = \lambda_k^* = 1/|S|$, $j \in S$, $k \in P$ is a feasible solution to *CP*.

We know, as highlighted in §2.2, that every solution to *CP* is minimal. Using the minimalistic property, and as proven in Elhallaoui et al. (2011), we can confirm that this solution to *CP* is unique, in a sense that x_j^*, $j \in S$ must take $1/|S|$ as value. □

PROPOSITION 7. *S corresponds to a nondecomposable set Q (Balas and Padberg 1975) that permits us to obtain x^{**}, a new lower-cost integer solution to $\mathbb{P}$ and $Q^- = \varnothing$.*

PROOF. We consider $\sum_{j\in S} \bar{A}_j^k$ to determine the variables that are modified in order to obtain a feasible solution when we set to 1 the variables with indices in S.

For $k \in N$

$$\sum_{j\in S} \bar{a}_{kj} = |S| \sum_{j\in S} \bar{a}_{kj} x_j^* = 0.$$

For $k \in P$, we have $\bar{a}_{kj} = a_{kj}$ by 5 and $\sum_{j\in S} a_{kj}$ is binary because the columns A_j, $j \in S$ are disjoint.

Let

$$S^+ = \left\{k \,\middle|\, \sum_{j\in S} \bar{a}_{kj} = 1\right\} \subseteq P, \quad Q^- = S^- = \left\{k \,\middle|\, \sum_{j\in S} \bar{a}_{kj} = -1\right\} = \varnothing.$$

S satisfies the condition (1)–(2) of Theorems 1 and 2 of Balas and Padberg (1975) and

$$x_j^{**} = \begin{cases} 1, & j \in S \cup S^- \cup (P - S^+) = S \cup (P - S^+) \\ 0, & \text{otherwise} \end{cases}$$

is a feasible integer solution to $\mathbb{P}$.

We now calculate the cost of solution x^{**}. By Proposition 4, we have

$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j\in S} c_j - c_B \sum_{j\in S} \bar{A}_j,$$

$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j\in S} c_j - c_P \sum_{j\in S} A_j^P,$$

because

$$\left\{k \,\middle|\, \sum_{j\in S} \bar{a}_{kj} \neq 0\right\} = S^+ \subseteq P \subseteq B, \quad \bar{A}_j^N = 0, \quad j \in S \quad \text{and}$$

$$\bar{A}_j^P = A_j^P, \quad j \in S.$$

$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j\in S} \bar{c}_j \quad \text{because } \bar{c}_j = c_j - c_P A_j^P$$

$$c \cdot x^{**} = c \cdot (x_c^*, 0) + |S| \cdot \bar{c}_I x_I^*.$$

Therefore,

$$c \cdot x^{**} < c \cdot (x_c^*, 0) \quad \text{because } \bar{c}_I \cdot x_I^* < 0.$$

Thus, we obtain a new lower-cost integer solution. □

REMARK 2. The sets S generated by the *CP* are a particularly interesting case of the sets Q of Balas and Padberg (1975).

Since $S^- = \varnothing$, we move from the integer solution $(x_c^*, 0)$ to the adjacent integer solution x^{**} by adding only the set S to the variables equal to 1. We do not set to 1 basic variables that were previously 0. The variables of S^+ change from 1 to 0 as in Balas and Padberg (1975).

3.3. Branching to Obtain Column-Disjoint Solutions to *CP*

When the solution to *CP* is not column disjoint, we use the following branching method. Let V be the columns of S that have a nonempty intersection with another column of S; $V = \{j \mid j \in S$, there exists $i \in S$ such that $A_i \cdot A_j \neq 0\}$. $V \neq \varnothing$ because S is not column disjoint. We define $|V| + 1$ branches, a branch 0 and a branch j for each $j \in V$:

Branch 0: $x_j = 0$ for $j \in S$. These constraints are easy to impose. It is sufficient to remove the variables set to zero from *CP*.

Branch j: $x_i = 0$ for $i \in I$, $i \neq j$ and $A_i \cdot A_j \neq 0$ and x_j free. These constraints are easy to impose. It is sufficient to remove variables set to zero from *CP*. It would be good to impose $x_j \neq 0$ in order to have disjoint branches but this is not easy. We can not impose $x_j = 1$ because in any solution to *CP* we have $x_j < 1$. We can not impose $x_j = 1/|S|$ because the size of S can change at the next solution of *CP*.

All these branches eliminate the current solution $x_j > 0$, $j \in S$ because we fix to zero at least one variable of S. When exploring the branches, the algorithm gives priority to the zero branch. The idea is that solutions to *CP* have a high probability of being column disjoint. If a solution is not, we look elsewhere where it probably is. If the solution to *CP* in branch 0 is not column disjoint, we branch again and explore the next zero branch. This deep search continues as long as the marginal cost of the solution of *CP* is negative. If we must explore the j branches to obtain a solution, we start with j having minimum $\bar{c}_j$.

It is not necessary to explore the full branching tree. As soon as a column-disjoint solution with a negative marginal cost is obtained in some branch, it can be added to *RP* to improve the current integer solution. The current tree is abandoned and a new tree is considered at the subsequent iteration.

3.4. Improving the Current Integer Solution by *CP* Is Sufficient to Reach Optimality

This section justifies the control step of the algorithm. It is sufficient to prove that Step 2 permits to reach optimality.

A pivot in Step 1 is a special case of Step 2 where the set Q reduces to a single column. Step 1 is a faster way to improve the integer solution when it is possible.

The following proposition shows that the sets Q with $Q^- = \varnothing$ are sufficient to move from an integer solution to $\mathbb{P}$ to an optimal solution to $\mathbb{P}$. For $i = 1, 2$, let x^i be an integer solution to $\mathbb{P}$, I_i and J_i the basic and nonbasic index sets, and $Q_i = \{j \mid x^i_j = 1\}$.

Proposition 8. *If x^1 is a nonoptimal integer solution to $\mathbb{P}$ and x^2 is a lower-cost integer solution to $\mathbb{P}$ then there exists $Q \subseteq J_1$ such that*

1. $Q^+ = \{k \mid \sum_{j\in Q} \bar{A}^k_j = 1\} \subseteq Q_1$,
2. $Q^- = \{k \mid \sum_{j\in Q} \bar{A}^k_j = -1\} = \varnothing$,
3. $Q_2 = Q \cup Q^- \cup (Q_1 - Q^+)$,
4. *Q is column disjoint.*

Proof. Let $Q = Q_2 - (Q_1 \cap Q_2)$. To study $Q^+ = \{k \mid \sum_{j\in Q_2-(Q_1\cap Q_2)} \bar{A}_j = 1\}$, we first prove the following lemma. □

Lemma 2. $$\sum_{j\in Q_1-(Q_1\cap Q_2)} \bar{A}_j = \sum_{j\in Q} \bar{A}_j.$$

Proof. First $\sum_{j\in Q_1} A_j = \sum_{j\in Q_2} A_j = e$ because x^1 and x^2 are solutions to $\mathbb{P}$.

We obtain by subtracting the A_j, $j \in Q_1 \cap Q_2$:

$$\sum_{j\in Q_1-(Q_1\cap Q_2)} A_j = \sum_{j\in Q_2-(Q_1\cap Q_2)} A_j = \sum_{j\in Q} A_j.$$

Multiplying by B^{-1}, the inverse of the basis B associated with x^1, we obtain

$$\sum_{j\in Q_1-(Q_1\cap Q_2)} \bar{A}_j = \sum_{j\in Q} \bar{A}_j. \quad \square$$

Proof of 1. We have

$$Q^+ = \left\{k \,\middle|\, \sum_{j\in Q_1-(Q_1\cap Q_2)} \bar{A}_j = 1\right\} = Q_1 - (Q_1 \cap Q_2) \subseteq Q_1$$

because

$\bar{A}_j = e_j$ for $j \in Q_1$, where e_j is a column of I.

Proof of 2. We have

$$Q^- = \left\{k \,\middle|\, \sum_{j\in Q} \bar{A}^k_j = -1\right\} = \left\{k \,\middle|\, \sum_{j\in Q_1-(Q_1\cap Q_2)} \bar{A}_j = -1\right\} = \varnothing$$

because

$\bar{A}_j = e_j$ for $j \in Q_1$.

Proof of 3. We have

$$Q \cup Q^- \cup (Q_1 - Q^+)$$
$$= (Q_2 - (Q_1 \cap Q_2)) \cup \varnothing \cup (Q_1 - (Q_1 - (Q_1 \cap Q_2)))$$

using the definition of Q and the values of Q^- and Q^+.

$$= (Q_2 - (Q_1 \cap Q_2)) \cup (Q_1 \cap Q_2) = Q_2.$$

Proof of 4. Q is column disjoint because $Q \subseteq Q_2$, which is a basic integer solution of the set partitioning problem.

Let us show that Q of Proposition 11 defines a column-disjoint solution to *CP* with a negative reduced cost.

Proposition 9. *If x^1 is a nonoptimal integer solution to $\mathbb{P}$, x^2 a lower-cost integer solution to $\mathbb{P}$, and Q the set defined in Proposition* 11, *then $\hat{x}_j = 1/|Q|$, $j \in Q$ and $\hat{x}_j = 0$, $j \notin Q$ is a feasible solution, with a negative reduced cost, to the complementary problem defined from x^1.*

Proof. For the solution x^1, Q_1 is the index set of the non-degenerate constraints and its complement N is the index set of the degenerate constraints:

$$\sum_{j\in Q} \bar{A}_j = \sum_{j\in Q_1-(Q_1\cap Q_2)} \bar{A}_j \quad \text{by Lemma 2.}$$
$$= \sum_{j\in Q_1-(Q_1\cap Q_2)} e_j \quad \text{because } \bar{A}_j = e_j,\ j \in Q_1.$$

Therefore,

$$\sum_{j\in Q} \bar{A}^N_j = \sum_{j\in Q_1-(Q_1\cap Q_2)} 0^N = 0 \quad \text{because } N \cap Q_1 = \varnothing.$$

Hence,

$$\sum_{j\in I} \bar{A}^N_j \hat{x}_j = \sum_{j\in Q} \bar{A}^N_j \hat{x}_j = \frac{1}{|Q|} \sum_{j\in Q} \bar{A}^N_j = 0.$$

The last constraint is also satisfied:

$$e \cdot \hat{x} = \sum_{j\in I} \hat{x}_j = \sum_{j\in Q} \hat{x}_j = |Q| \cdot \frac{1}{|Q|} = 1.$$

The reduced cost of this solution is $\sum_{j\in Q} \bar{c}_j \cdot \hat{x}_j = \sum_{j\in Q} \bar{c}_j (1/|Q|) = (1/|Q|) \sum_{j\in Q} \bar{c}_j$.

This value is calculated from the result of Proposition 4:

$$c \cdot x^2 = c \cdot x^1 + \sum_{j\in Q} c_j - c \cdot \sum_{j\in Q} \bar{A}_j.$$

Hence,

$$0 > cx^2 - cx^1 = \sum_{j\in Q} c_j - c_B \cdot \sum_{j\in Q} \bar{A}_j,$$
$$0 > \sum_{j\in Q} c_j - c_P \sum_{j\in Q} \bar{A}^P_j - c_N \cdot \sum_{j\in Q} \bar{A}^N_j$$

using the row partition (P, N), on the column.

In the basis, a column corresponds to each row:

$$0 > \sum_{j\in Q} c_j - c_P \sum_{j\in Q} A^P_j \quad \text{because } \sum_{j\in Q} \bar{A}^N_j = 0 \text{ and } \bar{A}^P_j = A^P_j,$$
$$0 > \sum_{j\in Q} \bar{c}_j.$$

The reduced cost of this solution is thus negative. □

Remark 3. The set Q from Propositions 11 and 12 can be decomposable. In this case, there are nondecomposable subsets Q_i of Q. Proposition 12 also applies to each set Q_i. Each Q_i defines a minimal feasible solution to the complementary problem.

4. Improvement of Balas and Padberg Results on the Sequence of Adjacent Bases Leading to Optimality

Balas and Padberg (1972, 1975) proved the existence of a sequence of adjacent bases of nonincreasing cost from an initial integer solution to a given better integer solution. Their sequence uses some pivots on negative $\bar{A}_j^k$. ISUD is a constructive approach producing a sequence without pivots on negative $\bar{A}_j^k$ and does not need to know a better a priori integer solution.

4.1. The Type of Pivots Used by the ISUD Algorithm

As discussed in §3, the *RP* can be solved with the simplex algorithm using regular pivots on positive $\bar{A}_j^k$. The pivots on negative $\bar{A}_j^k$ are not necessary with ISUD. In the other algorithms presented in the literature (Balas and Padberg 1972, 1975; Haus et al. 2001; Rönnberg and Larsson 2009; Thompson 2002), the basis contains zero variables. Degenerate pivots on negative $\bar{A}_j^k$ are used to exchange some zero variables in the basis with nonbasic variables. In ISUD, there are no zero basic variables, and these kinds of pivots are not necessary.

The *CP* can also be solved with ordinary simplex pivots. It is an ordinary linear programming problem embedded in a branching tree. Because ISUD uses only ordinary simplex pivots the algorithm can be implemented with a regular simplex algorithm program without any internal modification. We implemented it with CPLEX taking advantage of all the know-how embedded in this black box.

4.2. The Number of Pivots in ISUD

Balas and Padberg (1972, 1975) present a bound on the number of primal pivots in their sequences of adjacent bases between two integer solutions but the proof supposes the knowledge of the final solution. ISUD does not use this information and does not have the same strong result. It is possible to prove that ISUD does exactly $|S|$ pivots when adding a set S to improve the solution of *RP* (see Propositions EC.1 and EC.2 in the e-companion available at http://dx.doi.org/10.1287/opre.2013.1247). This proof is not for a practical use. Actually, ISUD uses a better and simple solution method to update the current integer solution (see §5).

When we solve *CP* with the primal simplex without knowing the set S in advance, more than $|S|$ pivots may be necessary to reach the optimal solution when the entry criterion is to choose the variable with the minimum reduced cost. For example, if there are several sets S_l defining feasible solutions of *CP*, we can enter into the basis variables from several S_l via degenerate pivots before to reach an optimal solution. Indeed, even if S_{l^*} has a minimal mean marginal cost (i.e., $\sum_{j\in S_{l^*}} \bar{c}_j x_j$ is minimal s.t. $\sum_{j\in S_{l^*}} x_j = 1$) this does not imply that the $\bar{c}_j$ of the set S_{l^*} are inferior to all the $\bar{c}_j$ of the set $I - S_{l^*}$.

The e-companion translates the sequence of basic solutions generated by ISUD into the original polyhedron and discusses the difference with the sequence of Balas and Padberg.

5. Implementation Tips

This section presents some efficient methods to solve the reduced problem and the complementary problem. For example, the multi-phase strategy explained below permits a significant speedup of the algorithm. The justifications in this section are based on intuitive discussions, observations, and numerical results presented in §6. Nothing in this section can infirm the validity of the algorithm. The effects are only on its speed.

The *RP* is a nondegenerate linear program. It can easily be solved with the primal simplex algorithm. Each pivot improves the solution. Furthermore, the pivots are regular pivots on positive $\bar{A}_j^k$. The implementation of ISUD performs better than the simplex algorithm in solving the *RP* by using the following remark.

Remark 4. In Step 1 and Step 2, modifications to the *RP* solution can be conducted without using simplex pivots.

Actually, updating the *RP* solution (that is also solution to the original problem) either by entering into the basis a single compatible variable x_j (in Step 1), (say $S = \{j\}$) or by many incompatible variables x_j, $j \in S$ (in Step 2), where S is associated with a solution to *CP*, is done as follows:

- the new solution is obtained by changing from 0 to 1, x_j, $j \in S$ and by changing from 1 to 0 the variable(s) covering the same constraints,
- the new basis is still I,
- the new dual variables are $\pi = c_B\ B^{-1} = c_B\ I = c_B$.

And because *CP* is severely degenerate primal simplex is not a good algorithm to solve it, the dual simplex is a more appropriate algorithm. The solution that sets to 1 the variable with the most negative reduced cost is a good starting dual feasible solution. Another method that is effective for this problem is the Presolve algorithm of CPLEX. It often reduces the *CP* problem by more than 90% and the solution to the residual problem with the dual is rapid. In §6, we present numerical experiments on the solution of the complementary problem.

The numerical results show that the set S associated with the optimal solution to *CP* is not only minimal but generally small. Indeed, for the solutions in the form identified in Proposition 9, the objective value is $(1/|S|)\sum_{j\in S} \bar{c}_j$. It is therefore equal to the average cost of the $\bar{c}_j$ in S. This cost is minimal if S contains the smallest $\bar{c}_j$ allowing the constraints to be satisfied. Increasing the set S leads to the addition of larger $\bar{c}_j$ values. We discuss the potential impact of the test problems properties on the size of sets S in the numerical results section (§6). Another formulation of *CP* has been considered; it replaces the constraint $\sum_{i\in I} x_i = 1$ by $x_i \leqslant 1$, $i \in I$. This formulation produces much larger

sets S and does not have the same good properties: S is larger and not minimal, the probability of being column disjoint is small, and the values of the positive variables can be different of $1/|S|$.

Small S sets are more likely to be column disjoint. Indeed, we will see in the numerical results that the optimal solutions of CP are often column disjoint even if this constraint is relaxed. The solution of CP is therefore an effective way to identify sets of columns that permit us to move from one integer solution to an adjacent integer solution of a lower cost.

The multi-phase strategy introduced by Elhallaoui et al. (2010) could be used here to accelerate solving CP. This strategy proceeds through a sequence of phases. Each phase corresponds to a different level of partial pricing. The level is defined by the number of incompatibilities that a column $j \in I$ can have with respect to the partition defined by the integer solution to RP. Let the cluster of this partition defined by the columns for which $x_l = 1$, $l = 1, \ldots, p$ be $W_l = \{k \mid A^k_l = 1\}$, $l = 1, \ldots, p$. This number can be mathematically defined as follows:

Definition 3. Given a partition $W_l = \{k \mid A^k_l = 1\}$, $l = 1, \ldots, p$ and a column j, the number of incompatibilities of j with respect to W is given by $k^W_j = \sum_{l=1}^{p} k_{lj}$, where k_{lj} is equal to 1 if column j covers some, but not all, of the elements in W_l, and 0 otherwise. A column j and its associated variable x_j are said to be *k-incompatible* with respect to W if $k^W_j = k$. Compatible columns and variables are also qualified as 0-incompatible variables.

A phase of the ISUD algorithm is defined as follows.

Definition 4. The ISUD algorithm is said to be *in phase* k when, among the variables x_j, only those that are q-incompatible with $q \leqslant k$ are priced out by CP. k is called *the phase number*.

The sequence of phases that the algorithm goes through is predetermined and, to ensure the exactness of the algorithm, it must terminate with a phase where all variables are priced out. Observe that p is an upper bound on k^W_j. The advantages of the multi-phase algorithm on the mono-phase algorithm are presented in the experimentation section.

The results in §6 are produced with the implementation tips described above.

6. Experimentation

We tested the ISUD algorithm on a 2.8 Ghz dual core Linux machine where a single processor is used. The solver used is CPLEX (version 12.0). In our experiments, we run both CPLEX and ISUD on different instances and compared their performances. On the ISUD side, we included results from its mono-phase and multi-phase versions. The mono-phase version refers to the version of ISUD where we do not use the multi-phase strategy.

This first experimentation presents results from a heuristic branching for CP. The branching is a deep search without backtracking; exploring only the zero branch. The exploration stops when the reduced cost of the solution becomes greater or equal to zero. This partial exploration produces optimal solutions in many cases and near optimal solutions in the majority of the other cases.

6.1. Instances

To test the performance of ISUD on problems of different size, we used three problems:

- sppaa01 (small): aircrew scheduling problem from OR-Lib ($\sim$ 800 constraints $\times$ 8,000 variables).
- vcs1,200 (medium): randomly generated bus and driver scheduling problem ($\sim$ 1,200 constraints $\times$ 130,000 variables).
- vcs1,600 (large): randomly generated bus and driver scheduling problem ($\sim$ 1,600 constraints $\times$ 500,000 variables).

The number of nonzeros per column is in average *nine* in sppaa01 and 40 in vcs1,200 and vcs1,600 instances. These numbers of nonzeros are typical for aircrew and bus driver scheduling problems. They do not increase with the number of constraints. They are related to the number of flights per pairing for aircrews and the number of pieces of work per duty for bus drivers. Thus, these problems become fairly sparse when the number of constraints increases. This should help obtain disjoint columns in CP solutions more frequently. The ISUD is well adapted to real-life large-scale vehicle and crew scheduling problems and probably would work better on larger problems.

ISUD needs an initial solution to start from. So, those problems are first solved using CPLEX for the small problem and GENCOL for the medium and large problems. GENCOL is a commercial software developed at the GERAD research center in Montreal and now owned by AD OPT Technologies, a division of KRONOS. GENCOL supports the column generation method and is widely used to solve vehicle and crew scheduling problems. For the set partitioning problem, a greedy heuristic produces infeasible solutions most of the time. We discuss at the end of §6.2 the results obtained when starting up the ISUD with such greedy poor solutions.

We developed a method to randomly perturb the optimal solution to obtain a feasible initial solution with a certain level of good primal information similar to initial solutions generally available in practice for crew scheduling problems. The columns of the perturbed solution are added to the problem with a high cost. This method permits to obtain many instances (many starting solutions) from an initial problem.

The motivation behind this perturbation method comes from two observations. Firstly, in crew scheduling problems, we observe that crews do not change their vehicles very often; their rotations have thus many parts in common with the vehicle routes. Secondly, for reoptimization of

crew or vehicle schedules during their operation we notice that reoptimized solutions deviate slightly from planned ones. Consequently, many consecutive tasks (flights, bus trips) on the initial *paths* (the vehicle or planned solution paths in aircrew or bus and crew scheduling problems described above) will remain grouped in an optimal solution. The variables are associated with paths (pilot or driver schedules) that are feasible with regards to a set of predefined rules. A path contains an ordered sequence of tasks and possibly other activities (breaks, rests, briefings, ...).

We can consider as a measure of good primal information the percentage of consecutive pairs of tasks in the initial solution that remain grouped in the optimal solution. An initial solution following the bus route as much as possible has generally 90% or more of good primal information. In fact, in an optimal solution a bus driver stays on the same bus over 90% of the relief points, i.e., locations where driver exchanges can occur. Similarly, an initial crew pairing solution following the aircraft route has a high percentage of good primal information. For the short and medium haul problems, crews operate approximately five flights a day. Generally, more than 50% of them stay on the same aircraft all the day. The others change the aircraft one or two times a day. Less than 5% changes the aircraft two times. At the end of their working day, 20% operate the same aircraft the following day. For a typical day of five flights, we have an average of 1.35 aircraft changes (0.8×1 at the end of the day, and $0.5 \times 0 + 0.45 \times 1 + 0.05 \times 2$ during the day). From that, we can conclude that an aircrew changes the aircraft (after a flight) 27% (1.35/5) of the time or less. Meaning that an initial solution following the aircraft routes has more that 73% of good primal information.

Below, we present a perturbation process of the optimal solution to create an initial solution with a high percentage of good primal information. The perturbation process of a solution consists in randomly selecting two of its columns and replacing them with two columns covering same tasks as the two original columns. The perturbation process chooses two tasks belonging to two different paths in the solution (set of paths) and connecting them to create a new path.

This process is repeated until the number of unchanged columns (belonging to an initial solution) goes below a certain percentage of the number of columns in the solution. For our tests, we set this number to 50%, 35%, and 20%. Also, the newly generated columns are added to the problem and given the maximum cost of all columns in the problem. The perturbed columns are often selected again for supplementary perturbations and this process can go far from the initial optimal solution. Note that the optimal solution columns are not removed from the problem. The percentage of good primal information in the initial solutions we start from is discussed in the following subsection.

6.2. Results

Table 1 shows the mono-phase version of ISUD and CPLEX results from a set of 10 executions on different instances of the small problem (perturbed to 50%). We read from left to right: the percentage of the optimal solution columns left in the initial solution, the ISUD execution time, the CPLEX execution time, the time ratio comparing the performance of the two algorithms, the initial ISUD error on the optimal objective value (a percentage expressing how far is the optimal value from the one ISUD starts from), the final ISUD error, the number of *CPs* solved by ISUD, the number of the ones that had disjoint solutions among them, the maximum size of the disjoint solutions and their average. The bottom line contains column averages.

The mono-phase version was able to find disjoint solutions in 73% (on average) of the cases and succeeds to reach optimality in seven instances over 10. However, CPLEX was faster than the mono-phase version for this small problem. In fact, we include all the incompatible columns in *CP*. So, the *CP* time increases significantly and *CP* may select large sets of nondisjoint columns as candidates to improve the current solution. This means that strategies that help reducing *CP* time (like the multi-phase strategy) are needed to efficiently solve set partitioning problems. In three instances, the mono-phase version failed to reach optimality and the error was high. Actually, since the branching implemented is deep only, columns that would possibly be part of a better disjoint solution could be eliminated. A more sophisticated branching may reduce such errors.

Table 2 shows results for the multi-phase version of ISUD and CPLEX on 10 instances of the small problem for each level of perturbation. Two new columns are added to this table: the time ISUD reached its optimal solution (Opt.) and the phase in which ISUD finds its best solution (Phase). In this table, we observe that the multi-phase version is faster than CPLEX in most cases (and obviously largely faster than the mono-phase version). It reaches optimal solutions even faster in nine cases over 10. Actually, it produces solutions with less error on the objective value than the mono-phase version. Interestingly, it presents a higher ratio of disjoint solutions (81%) than the mono-phase version; which means it, more often, finds disjoint solutions without a need to branch. Observe that the solution time increases with the perturbation level of the initial solutions.

Figure 1 shows how fast each algorithm (mono-phase ISUD, multi-phase ISUD, CPLEX) reaches an optimal solution on instance 1 of the small problem perturbed to 50%. The multi-phase version of ISUD is definitely faster than the mono-phase version. Even if it solves more complementary problems than the mono-phase version, those problems are way smaller. In fact, it makes more iterations with small improvements to the initial solution but in a very short time. In this example, CPLEX is the fastest. Figure 2

Table 1. Mono-phase ISUD vs. CPLEX (small instance).

	Time (sec.)			Objective value (%)		*CP* solutions			
Orig. col. (%)	ISUD	CPLEX	Ratio %	Init. err.	Err.	Total	Disj.	Max	Avg.
50	27	5	540	360.6	0	15	13	17	3.9
	24	17	141.2	357.4	0	14	11	13	4.7
	30	13	230.8	347.7	0	17	15	11	3.5
	28	16	175	356.8	202.3	20	9	7	2.7
	25	14	178.6	353.3	0	17	15	13	3.4
	24	16	150	352.8	0	13	11	15	4.8
	32	15	213.3	364.2	0	17	15	9	3.5
	14	17	82.4	368	311.9	10	3	2	2
	28	16	175	360.1	0	15	12	14	4.2
	10	15	66.7	366.9	276.1	8	3	8	4
	24.2	14.4	195.3	358.78	79.03	14.6	10.7	10.9	3.67

Table 2. Multi-phase ISUD vs. CPLEX (small instance).

	Time (sec.)				Objective value (%)		*CP* solutions				
Orig. col. (%)	ISUD	Opt.	CPLEX	Ratio %	Init. err.	Err.	Phase	Total	Disj.	Max	Avg.
50	12	6	5	240	360.6	0	4	35	29	6	2.4
	9	4	17	52.9	357.4	0	3	38	31	7	2.4
	9	5	13	69.2	347.7	0	4	33	27	4	2.4
	13	10	15	86.7	356.8	0.1	5	56	39	8	2.9
	9	5	13	69.2	353.3	0	4	33	27	4	2.4
	8	4	16	50	352.8	0	4	33	27	5	2.6
	8	4	14	57.1	364.2	0	4	38	32	5	2.2
	10	6	18	55.6	368	0	4	41	33	9	2.9
	11	7	16	68.8	360.1	0	5	32	26	7	2.5
	10	5	15	66.7	366.9	0	4	36	31	10	2.5
	9.9	5.6	14.2	81.62	358.78	0.01	4.1	37.5	30.2	6.5	2.52
35	12	8	13	92.3	472.1	0	5	42	33	7	3
	13	8	16	81.2	472.8	0	6	39	29	19	3.4
	9	4	18	50	464.7	0	3	37	31	7	2.6
	10	6	9	111.1	468.4	0	5	35	27	17	3.1
	10	6	14	71.4	463.5	0	4	37	32	5	2.8
	11	8	15	73.3	466.3	0	5	50	39	8	2.8
	16	13	18	88.9	456.3	0	5	43	36	10	3.5
	10	5	15	66.7	471.6	0	3	46	39	5	2.3
	12	8	13	92.3	469.2	0.6	4	46	37	10	3.1
	9	6	7	128.6	464.6	0	5	40	30	7	2.7
	11.2	7.2	13.8	85.58	466.95	0.06	4.5	41.5	33.3	9.5	2.93
20	12	9	14	85.7	570.2	0	5	44	40	15	3.3
	19	15	18	105.6	561	0	5	53	44	8	3.1
	13	8	19	68.4	559.9	138.2	5	44	23	18	4.3
	14	10	7	200	557.7	0	4	48	43	13	3.3
	18	14	17	105.9	562.5	0	5	52	43	13	3.6
	14	10	14	100	561	0	5	42	33	12	3.4
	12	7	10	120	573.1	0	4	47	40	8	2.8
	16	12	19	84.2	569.6	0	6	51	40	8	3.2
	17	12	17	100	569.3	0	4	63	51	14	3.3
	9	4	13	69.2	573.7	200.6	4	43	31	11	3.5
	14.4	10.1	14.8	103.9	565.8	33.88	4.7	48.7	38.8	12	3.38

represents multi-phase ISUD and CPLEX on instance 4 for the small problem perturbed to 50%. The multi-phase version is faster than CPLEX but not optimal because of the partial branching. This version of ISUD finds many integer solutions that are close to each other. This property is good when using heuristic stopping criteria in large-scale problems.

Tables 3–4 show the same information collected on executions of the multi-phase version of ISUD, which we refer to hereunder by ISUD (without mentioning that is mutiphase), on the medium and large instances. The tables do not show any CPLEX execution because the latter was not able to find any integer solution in a reasonable time. In fact, we tried CPLEX to solve six samples

Figure 1. Mono-phase ISUD vs. multi-phase ISUD vs. CPLEX (small problem, instance 1, 50%).

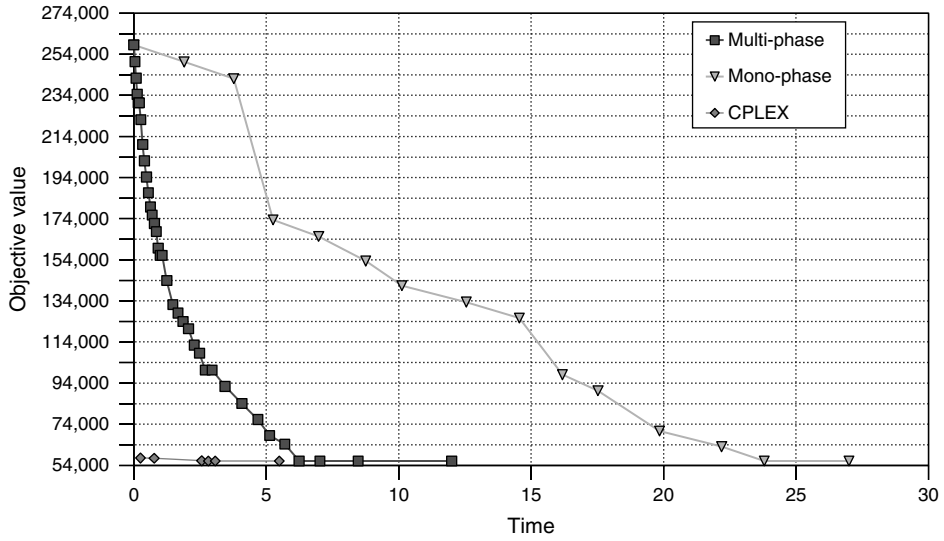
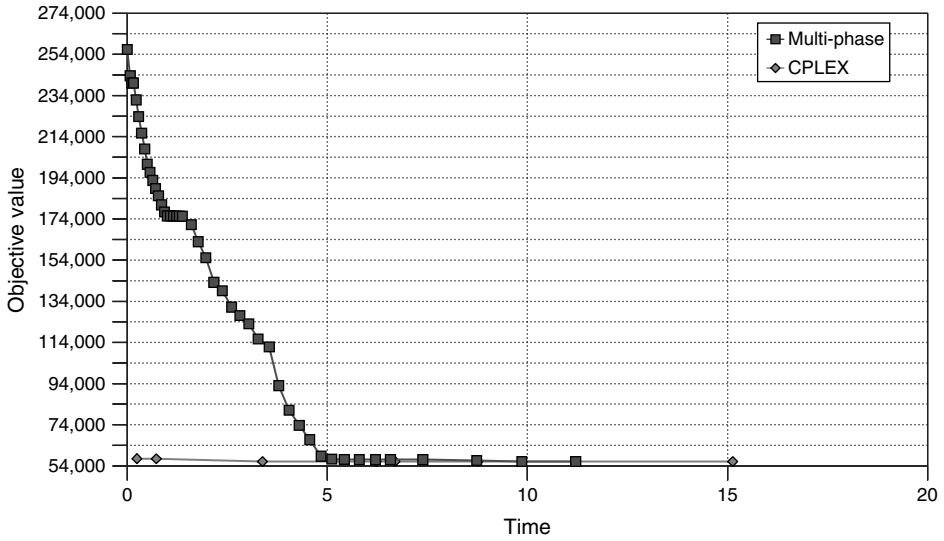


Figure 2. Multi-phase ISUD vs. CPLEX (small problem, instance 4, 50%).



(three instances of the medium problem and three of the large one). All the tests were aborted after 10 hours of CPLEX runtime. CPLEX showed an average of 69.55 seconds on the LP relaxation solution time for the medium instances and 592.25 seconds for the large ones. Within the time frame of 10 hours, not even a single integer solution was encountered for all the samples. Interestingly, *CP* produces relatively small convex combinations with size varying between 2 and 13. These convex combinations are often column disjoint (89%). The ISUD results are very good for both medium and large instances. When the initial solution is far from the optimal solution, the quality of ISUD solutions decreases and the computation time increases. Developing smart heuristics to find better initial solution will help reducing ISUD time especially for large instances.

The mutli-phase version of ISUD outperforms CPLEX in most test cases. They behave in the same way for small test instances with a slight advantage for ISUD, but for large instances, ISUD solves in few minutes what the CPLEX cannot solve.

We can say that ISUD is particularly useful when the size of the problem is large. As a whole, we conclude the following:

- ISUD reaches optimal solutions most of the time.
- ISUD is faster than CPLEX.
- ISUD is relatively stable compared to CPLEX.
- The quality of ISUD solutions is better for large problems (smaller error).
- The solution time increases with the perturbation level of the initial solutions. It profits from a good heuristic solution if available.

Table 3. ISUD results for medium instances.

	Time (sec.)		Objective value (%)		*CP* solutions				
Orig. col. (%)	ISUD	Opt.	Init. err.	Err.	Phase	Total	Disj.	Max	Avg.
50	59	24	53.43	0	4	15	14	3	2.1
	89	55	51.39	0	5	17	16	3	2.1
	81	51	53.43	0	5	16	15	3	2.2
	126	55	53.44	0	5	15	14	4	2.4
	145	93	51.38	0	6	15	12	6	2.4
	63	32	51.38	0	5	16	15	3	2.1
	165	107	51.38	0	6	18	16	3	2.1
	264	110	51.38	0	6	18	16	3	2.1
	200	102	51.38	0	7	19	15	5	2.3
	57	21	51.38	0	4	15	14	3	2.1
	124.9	65	52.00	0	5.3	16.4	14.7	3.6	2.19
35	221	165	65.77	0	6	24	21	4	2.3
	98	68	65.77	0	5	18	17	4	2.3
	147	80	65.77	10.27	6	19	15	3	2.3
	308	128	67.83	12.33	7	22	17	3	2.2
	360	191	65.77	0	7	23	19	5	2.5
	159	107	67.83	0	6	22	20	6	2.3
	324	222	67.82	0	7	24	20	4	2.4
	114	60	65.77	0	6	16	14	8	2.7
	167	113	65.77	0	6	22	19	5	2.3
	167	110	67.83	0	6	22	20	3	2.2
	206.5	124.4	66.59	2.26	6.2	21.2	18.2	4.5	2.35
20	397	230	82.21	0	8	22	17	8	3.2
	268	201	82.21	0	6	26	24	4	2.6
	142	105	82.21	0	5	25	24	5	2.5
	241	176	82.21	0	6	25	23	5	2.7
	89	64	82.21	41.11	6	15	15	4	2.5
	38	17	82.21	63.72	6	8	8	3	2.1
	383	215	82.21	0	8	27	21	5	2.6
	243	146	82.21	22.61	6	21	18	4	2.6
	298	124	82.21	0	6	23	21	5	2.6
	114	55	82.21	32.88	6	16	15	5	2.4
	221.3	133.3	82.21	16.03	6.3	20.8	18.6	4.8	2.58

• *CP* produces relatively small and often disjoint convex combinations with size varying between 2 and 19.

These good results were obtained on airline crew pairing problems and on bus driver scheduling problems. These are two important domains where set partitioning problems are utilized by the industry. In these domains, it is easy to obtain an initial solution containing good primal information.

Table 5 presents some information on the initial solution of the first problem of each group. We observe that the percentage good primal information is similar to what is available for real-life crew scheduling problems. The last two columns of Table 5 give the maximum and the average degree of incompatibilities of the columns. These large numbers prove that the same columns were perturbed many times. The number of perturbations grows with the size of the problem. We needed to perturb more before reaching a desirable percentage of unperturbed columns.

We also experiment with a greedy solution for the small problem. This solution has only 29% of good primal information and 37% of the flights are covered with artificial variables. The ISUD improved 46 times the greedy solution, i.e., ISUD found a decreasing sequence of 46 integer solutions, before stopping and reduced to 7% the number of flights covered by artificial variables. The mean cost per column (excluding the artificial ones) was 35% below in the greedy than the mean cost per column in the optimal solution. The greedy solution contains columns good with regards to the dual feasibility but very poor in primal information. It is the opposite of what ISUD likes to exploit.

It is also possible to have good primal information for vehicle, crew, and many other personnel scheduling problems when we reoptimize a planned solution after some perturbations. A large part of the planned solution will remain in the reoptimized solution. A similar situation appears in two-stage stochastic programming solved with the L-shaped method. For each scenario, a subproblem updates the planned solution according to the modified data. A large part of the planned solution remains in the solution of each subproblem.

Even if the conclusion of the experimentation cannot yet be generalized to all set partitioning problems, it is already applicable to a wide class of problems very important in practice. For the other problems, one could see how to

Table 4. ISUD results for large instances.

	Time (sec.)		Objective value (%)		*CP* solutions				
Orig. col. (%)	ISUD	Opt.	Init. err.	Err.	Phase	Total	Disj.	Max	Avg.
50	996	461	51.88	0	6	18	18	4	2.4
	1,173	542	50.34	0	6	19	19	4	2.2
	473	194	50.34	0	5	17	17	4	2.2
	303	198	50.35	13.73	5	15	15	2	2
	1,815	520	50.34	0	7	16	16	4	2.5
	392	183	51.87	0	5	19	19	3	2.1
	725	323	50.34	10.68	7	16	13	4	2.3
	1,050	376	50.35	0	6	17	17	4	2.2
	918	321	51.87	0	6	16	16	4	2.4
	2,887	648	50.34	0	7	19	19	3	2.2
	1,073.2	376.6	50.80	2.44	6	17.2	16.9	3.6	2.25
35	1,258	632	65.60	0	6	25	25	3	2.2
	1,166	557	65.60	0	6	23	23	5	2.5
	1,102	438	65.59	0	6	24	24	5	2.3
	2,655	765	65.60	0	7	26	26	5	2.3
	623	351	65.60	15.26	6	23	21	5	2.3
	2,061	813	65.60	0	7	24	24	5	2.4
	460	268	65.60	19.83	5	18	17	5	2.5
	517	419	65.60	6.10	5	24	24	5	2.3
	2,691	1,191	65.59	0	8	27	25	4	2.6
	928	436	65.60	10.68	6	23	22	7	2.5
	1,346.1	587	65.60	5.19	6.2	23.7	23.1	4.9	2.39
20	2,806	1,402	82.38	0	7	27	27	7	3.4
	2,968	1,194	80.85	0	8	24	22	13	3.7
	2,472	1,029	80.85	0	7	31	30	7	2.5
	2,363	1,120	80.86	0	8	23	21	13	3.4
	3,556	1,981	80.86	0	8	30	27	8	3.3
	1,553	1,076	80.86	0	6	32	32	6	2.6
	229	138	80.85	56.45	6	12	12	4	2.3
	3,293	1,845	80.85	0	7	31	31	8	3.2
	2,008	842	80.86	0	8	21	19	10	3.6
	2,923	1,247	80.86	0	7	34	33	4	2.6
	2,417.1	1,187.4	81.01	5.64	7.2	26.5	25.4	8	3.06

Table 5. Information on initial solutions.

Problem	Orig. col. %	% of good primal information	Max. degree of incompatibility	Avg. degree of incompatibility
Small	50	88	7	1.2
(airline)	35	82	9	2
	20	73	12	3
Medium	50	94	17	2.6
(bus)	35	90	17	4.5
	20	88	17	5.3
Large	50	91	12	2.2
(bus)	35	83	15	4.1
	20	73	22	6.8

adapt the multi-phase strategy by modifying the way we compute the degree of incompatibilities.

7. Conclusion

We introduce a constructive method that finds a decreasing sequence of integer solutions to a set partitioning problem by decomposing it into a reduced problem *RP* and a complementary problem *CP*.

The optimization of *RP* with the primal simplex involves carrying out pivots on variables that move from one integer solution to a better one. When the *RP* has no more improving pivots, the *CP* identifies a group of variables producing a sequence of pivots moving to a better integer solution after some degenerate pivots. Iterations on *RP* and *CP* permit to reach an optimal integer solution by only using normal pivots on positive coefficients.

A first implementation of the algorithm with only a partial branching procedure produces optimal solutions in many cases and near optimal solutions in the majority of the other cases. On large scale problems, the very good solutions are obtained in a small fraction of the time to obtain same quality integer solutions with the LP relaxation and the branch and bound approach. In fact, when we relax the disjoint columns condition, the *CP* becomes a linear program to solve. This relaxed problem produces disjoint columns most of the time and permits to rapidly improve the integer solution. This opens up a new way to obtain very good integer solutions for large set partitioning problems.

Future research on branching strategies and/or cutting plane methods should be able to close the small optimality gap and produce a new optimal solution method. Many good branching/cutting methods have been proposed in the literature for the set partitioning problem. Some of these methods could probably be adapted to the $RP-CP$ decomposition of ISUD. Also, combining ISUD and metaheuristics to find a good initial solution and to well populate CP could reduce significantly the solution time and produce solutions with higher quality.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/opre.2013.1247.

References

Balas E, Padberg MW (1972) On the set-covering problem. *Oper. Res.* 20(6):1152–1161.

Balas E, Padberg M (1975) On the set-covering problem: II. An algorithm for set partitioning. *Oper. Res.* 23(1):74–90.

Elhallaoui I, Metrane A, Desaulniers G, Soumis F (2011) An improved primal simplex algorithm for degenerate linear programs. *INFORMS J. Comput.* 23(4):569–577.

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

Haus U-U, Köppe M, Weismantel R (2001) The integral basis method for integer programming. *Math. Methods Oper. Res.* 53(3):353–361.

Metrane A, Soumis F, Elhallaoui I (2010) Column generation decomposition with the degenerate constraints in the subproblem. *Eur. J. Oper. Res.* 207(1):37–44.

Raymond V, Soumis F, Orban D (2010) A new version of the improved primal simplex for degenerate linear programs. *Comput. Oper. Res.* 37(1):91–98.

Rönnberg E, Larsson T (2009) Column generation in the integral simplex method. *Eur. J. Oper. Res.* 192(1):333–342.

Saxena A (2003) Set-partitioning via integral simplex method. Ph.D. thesis, Carnegie Mellon University, Pittsburgh.

Thompson GL (2002) An integral simplex algorithm for solving combinatorial optimization problems. *Comput. Optim. Appl.* 22(3):351–367.

Abdelouahab Zaghrouti is a Ph.D. candidate at Polytechnique Montréal. He is mostly interested in developing solvers for hard combinatorial problems.

François Soumis joined Polytechnique Montréal in 1984. He has held the NSERC/Canadian Research Chair on Optimization of Large Transportation Systems since 2001. He is cofounder of the company AD OPT (acquired by Kronos) and cooperates with GIRO and many other transportation companies. He conducted many research projects with these companies, commercializing optimization systems for vehicle and crew scheduling. Many bus, airline, and rail networks in the world use these commercial systems.

Issmail El Hallaoui joined Polytechnique Montréal in 2011. His current research interests are focused on integer programming, degeneracy in linear and nonlinear programming, robust optimization, nondifferentiable convex optimization, vehicle routing, and crew scheduling.