Les Cahiers du GERAD ISSN: 0711–2440

Integral simplex using double decomposition

O. Foutlane, I. El Hallaoui,
P. Hansen

G–2017–73

September 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 simplex using double decomposition

Omar Foutlane [a, b]
Issmail El Hallaoui [a, b]
Pierre Hansen [a, c]

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

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

[c] *Department of Decision Sciences, HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

omar.foutlane@gerad.ca
issmail.elhallaoui@gerad.ca
pierre.hansen@gerad.ca

September 2017

Les Cahiers du GERAD
G–2017–73

Abstract: Given an integer solution, the integral simplex using decomposition (ISUD) seeks a descent direction that leads to an improved adjacent integer solution. It uses a horizontal decomposition (of a linear transformation of the constraint matrix). We propose an integral simplex using double decomposition (ISU2D). It uses an innovative disjoint vertical decomposition to find in parallel orthogonal descent directions leading to an integer solution with a larger improvement. Each descent direction identifies a set of variables that will leave the current solution and a set of entering variables with better costs. To find these directions, we develop a dynamic decomposition approach that splits the original problem into subproblems that are then solved in parallel by ISUD. Our main innovation is the use of the current solution as a foundation for the construction of the set of subproblems; the set changes during the optimization process as the current solution changes. In addition, we use bounding and pricing strategies and implement parallel processing techniques. We show that ISU2D is faster than ISUD: 3 to 4 times faster on large instances.

Keywords: Set partitioning problems, integral simplex using decomposition, decomposition, parallel computing

1 Introduction

The set partitioning problem (SPP) is often used to model real-world combinatorial optimization problems including vehicle and crew scheduling problems. We use scheduling terminology to present the problem. A set partitioning constraint ensures that a *task* (for example, a flight leg or bus trip) is performed exactly once by a crew member (a pilot or bus driver). Let $T = \{1, 2, ..., m\}$ be the set of tasks and $J = \{1, 2, ..., n\}$ the set of feasible schedules. Here *feasible* means that the schedules satisfy all the safety and collective agreement rules limiting, for example, the maximum flying time during a working day and the maximum time away from the base. With each schedule j, we associate a variable x_j, a cost c_j, and a column $Aj = (a_{tj})_{t \in T}$ where a_{tj} is 1 if A_j covers task t and 0 otherwise. The matrix $A = [A_1, A_2, ..., A_n]$ is a binary matrix. The SPP formulation is:

$$(SPP) \quad \begin{aligned} & Minimize && \sum_{j \in J} c_j x_j && (1) \\ & subject\ to && && \\ & && \sum_{j \in J} a_{tj} x_j = 1, \forall t \in T && (2) \\ & && x_j \in \{0, 1\}, \forall j \in J && (3) \end{aligned}$$

The objective function (1) minimizes the total cost. The set partitioning constraints (2) ensure that each task is covered exactly once. Constraints set (3) imposes integrality on the x_j variables. The linear relaxation LP is obtained by replacing (3) by $x_j \geq 0, \forall j \in J$. The reduced cost of variable x_j, with respect to a dual vector dictated by the context, is denoted $\bar{c}_j$.

1.1 Literature review

The SPP is NP-hard (Garey and Johnson 1979). A partial list of its applications includes truck deliveries (Balinski and Quandt 1964), vehicle scheduling (Ribeiro and Soumis 1991), aircrew and bus driver scheduling (Desaulniers et al. 1994, Chu et al. 1997, Hoffman and Padberg 1993), and clustering and classification (Rao 1971). Many SPP algorithms have been developed. They can be classified into two main families. Dual methods (called ***dual-fractional*** in Letchford and Lodi (2002)) include the famous branch and cut method (see, e.g., Hoffman and Padberg 1993, Desaulniers et al. 1997). These methods are efficient for small and medium problems but less efficient for large SPPs. They may take days to find good solutions for some aircrew scheduling problems. They do not take advantage of available primal information, as we will explain later.

Primal methods move from an integer solution to a better one. Many primal methods are based on the famous result (Balas and Padberg 1975) that demonstrated the existence of a sequence of integer solutions with decreasing costs leading to an optimal integer solution; see Haus et al. (2001), Thompson (2002), Saxena (2003), and Rönnberg and Larsson (2009). Unfortunately, these algorithms suffer from degeneracy and are not efficient for large SPPs. Zaghrouti et al. (2014) developed the integral simplex using decomposition (ISUD), which is based on the improved primal simplex (IPS) decomposition introduced by El Hallaoui et al. (2010) to handle degeneracy. ISUD decomposes a linear transformation of the constraint matrix horizontally. The first group of constraints is handled in a reduced problem and the second group in the so-called complementary problem. ISUD handles degeneracy efficiently and is able to solve problems with up to 570000 columns and 1600 constraints.

Many authors have developed parallel algorithms based on the dual-fractional paradigm to take advantage of the availability of inexpensive parallel machines. Some of these explore the branching tree in parallel. For instance, Eso (1999) proposed a parallel branch-and-cut solver; Klabjan et al. (2001) and Alefragis et al. (1999) presented parallel algorithms for crew scheduling problems; and Linderoth et al. (2001) developed a parallel heuristic. Other parallel algorithms use domain decomposition techniques to split the original problem into subproblems. The subproblems are solved in parallel, and the solutions are merged to form a solution to the original problem. Topaloglu and Powell (2005) proposed a parallel heuristic with both time and space decomposition for a resource allocation problem. Abbink et al. (2007) studied various decompositions, such

as geographical, weekday, and line-based decompositions, for a Netherlands railway crew scheduling problem. They implemented a multi-stage method where at each stage they use a different decomposition. Jütte and Thonemann (2012) proposed a penalized-geographical decomposition for a railway crew scheduling problem. They decomposed the problem into overlapping regions that are optimized in parallel. The objective function penalized misclassified railway stations to adjust the region boundaries during the optimization process.

All these methods are heuristic and based on prior knowledge of the problem. They handle very large SPPs but do not guarantee optimality. Moreover, they are generally static with no way to adjust the decomposition during the solution process. The exception is the penalized-geographical decomposition method (Jütte and Thonemann 2012), where the modeler must identify the region boundaries at the beginning of the optimization process.

1.2 Contributions and organization

To the best of our knowledge, there are currently no parallel primal algorithms. We present a parallel primal algorithm called the *integral simplex using double decomposition (ISU2D)*, based on ISUD, that can solve large SPPs more efficiently. Given an integer solution, ISUD seeks a descent direction that leads to an improved adjacent integer solution. It uses a horizontal decomposition of a linear transformation of the constraint matrix. In addition to this horizontal decomposition, ISU2D uses disjoint and incremental vertical decompositions.

ISU2D uses an innovative disjoint vertical decomposition to find in parallel orthogonal descent directions leading to an integer solution with a larger improvement. Each descent direction identifies a set of variables that will leave the current solution and a set of entering variables with better costs. To find these directions, we develop a dynamic decomposition approach that splits the original problem into subproblems that are then solved in parallel by ISUD. Our main innovation is the use of the current solution as a foundation for the construction of the set of subproblems; the set changes during the optimization process as the current solution changes. We construct a weighted graph where the nodes represent variables at the value 1 in the current solution, and the edge weights are computed as a function of the reduced costs and other relevant information on the variables at the value 0. We divide the resulting graph into subgraphs where each subgraph is used to define a subproblem. This approach minimizes the role of the modeler and can be applied to a wide range of SPPs. Furthermore, we compute in parallel a lower bound to assess the solution quality during the optimization process. Using this lower bound and the dual information, ISU2D performs an incremental vertical decomposition to improve the integer solution obtained by the disjoint vertical decomposition. In the incremental decomposition, we identify a subset of variables that potentially improve the solution, and we add to this subset while the solution quality is unsatisfactory. In summary, we develop an enhanced algorithm that reduces the computational time of ISUD by a factor of three on average.

The remainder of this paper is organized as follows. Section 2 presents the theoretical and computational aspects of ISUD. Section 3 discusses ISU2D, and Section 4 presents the computational results demonstrating the effectiveness of ISU2D. Section 5 provides concluding remarks and suggestions for future research.

2 Preliminaries

In this section, we describe the ISUD algorithm and explain its main components. We discuss its limitations and propose the enhancements that lead to ISU2D.

2.1 ISUD overview

Given an integer solution $\bar{x}$ to the SPP, let P be the index set of its positive components, i.e., $P = supp(\bar{x}) = \{j \in J : \bar{x}_j = 1\}$, and $p = card(P)$. ISUD is a two-stage sequential algorithm that is specialized for the SPP. It is based on the concept of compatibility (El Hallaoui et al. 2010):

Definition 1 *A subset S of J is said to be compatible with an integer solution $\bar{x}$, or simply compatible, if there exist two vectors $v \in \mathbb{R}_+^{|S|}$ and $\lambda \in \mathbb{R}^p$ such that $\sum_{j\in S} v_j A_j = \sum_{l\in P} \lambda_l A_l$. The columns/variables indexed by S and the combination $\sum_{j\in S} v_j A_j$ are also said to be compatible. S is said to be minimal if any strict subset of it is incompatible.*

At the first stage, ISUD seeks compatible columns to improve $\bar{x}$. This is done by solving a reduced problem (RP), as explained in Section 2.2.1. The second stage looks for a compatible combination of (incompatible) columns that improves the $\bar{x}$ obtained at the first stage. It solves a complementary problem (CP) to find a descent direction d_{cp} leading to an improved integer solution. See Section 2.2 for the details of the decomposition. ISUD iterates between the two stages until it reaches an optimal solution. Algorithm 1 (Zaghrouti et al. 2014) outlines the procedure.

Algorithm 1 ISUD algorithm

Start with an initial integer solution x_0 and set $\bar{x} = x_0$

Stage RP: Improve the current solution $\bar{x}$ by solving RP.

Stage CP: Solve CP to get a descent direction d_{cp}.

IF $d_{cp} \neq 0$, i.e., CP improves the RP solution, THEN set $\bar{x} = \bar{x} + d_{cp}$ and go to Stage RP.

ELSE the integer solution $\bar{x}$ is optimal.

ENDIF

2.2 ISUD decomposition

We use the notation of Zaghrouti et al. (2014). Let $K \subset J$ and $L \subset T$. Let v_K (v^L) be the subvector of a vector v with components indexed in K (L). Similarly, $A_K^L = (a_{lk})_{l\in L, k\in K}$ is the $|L| \times |K|$ submatrix of A with rows and columns indexed by L and K respectively. If $L = T$ or $K = J$, the superscripts (subscripts) are omitted ($A_J^T = A$ for instance). Finally, e is a vector of ones with dimension dictated by the context, and I_K^K is the identity matrix of dimension $|K| \times |K|$.

We can permute without loss of generality the columns of A in such a way that its first p columns are those indexed in P. Zaghrouti et al. (2014) associate with $\bar{x}$ a basis B where the first p columns are those indexed in P and the remaining $m - p$ columns are artificial with a large cost. Let $N = T \setminus P$. We have

$$B = \begin{bmatrix} I_P^P & 0 \\ A_P^N & I_N^N \end{bmatrix} \text{ and } B^{-1} = \begin{bmatrix} I_P^P & 0 \\ -A_P^N & I_N^N \end{bmatrix}$$

because

$$\begin{bmatrix} I_P^P & 0 \\ -A_P^N & I_N^N \end{bmatrix} \begin{bmatrix} I_P^P & 0 \\ A_P^N & I_N^N \end{bmatrix} = \begin{bmatrix} I_P^P & 0 \\ 0 & I_N^N \end{bmatrix}.$$

When we multiply by B^{-1}, the constraint $Ax = e$ becomes

$$B^{-1}Ax = B^{-1}e \Leftrightarrow \bar{A}x = \bar{e} \Leftrightarrow \begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} [x] = \begin{bmatrix} \bar{e}^P \\ \bar{e}^N \end{bmatrix},$$

where the j^{th} column $\bar{A}_j$ is

$$\begin{bmatrix} \bar{A}_j^P \\ \bar{A}_j^N \end{bmatrix} = \begin{bmatrix} A_j^P \\ -A_P^N A_j^P + A_j^N \end{bmatrix}, \text{ and } \begin{bmatrix} \bar{e}^P \\ \bar{e}^N \end{bmatrix} = \begin{bmatrix} I_P^P & 0 \\ -A_P^N & I_N^N \end{bmatrix} \begin{bmatrix} e^P \\ e^N \end{bmatrix} = \begin{bmatrix} e^P \\ 0 \end{bmatrix}.$$

El Hallaoui et al. (2010) show that a column A_j is compatible iff $\bar{A}_j^N = 0$. Let C and I be the index sets of the compatible and incompatible columns. Thus, the set of columns J is partitioned into C and I, and the set of constraints T is partitioned into P and N. Hence, we write $x = [x_C, x_I]$, $A = [A_C, A_I]$, and $c = [c_C, c_I]$. Using this partition, Zaghrouti et al. (2014) decompose the problem as explained below.

2.2.1 Reduced problem

The reduced problem RP is defined by imposing $x_I = 0$, i.e., including compatible columns only:

$$
\begin{aligned}
& & Minimize \quad & c_C \cdot x_C && (4)\\
(RP) & & subject\ to \quad & \bar{A}^P_C x_C = e^P && (5)\\
& & & x_C \in \{0,1\}^{|C|} && (6)
\end{aligned}
$$

By definition, RP depends on $\bar{x}$. Zaghrouti et al. (2014) show that a pivot on any compatible column with a negative reduced cost leads to an improved integer solution. Let x^*_C be an optimal solution to RP. Note that $\bar{x} = (x^*_C, 0)$ is a solution to the SPP.

2.2.2 Complementary problem

Let $\bar{x}$ be the integer solution to the RP. ISUD solves a complementary problem CP to find a set of incompatible columns that improve $\bar{x}$. If we pivot on the columns in this set in any order we will obtain an improved integer solution. More precisely, we look for a set such that a (convex) combination of its columns is compatible and has a negative reduced cost. In other words, CP searches for a descent direction d_{cp} leading to an improved integer solution. Zaghrouti et al. (2014) formulate CP as follows:

$$
\begin{aligned}
& & Minimize \quad & \bar{c}_I \cdot x_I && (7)\\
(CP) & & subject\ to \quad & \bar{A}^N_I x_I = 0 && (8)\\
& & & e \cdot x_I = 1 && (9)\\
& & & x_I \geq 0 && (10)
\end{aligned}
$$

where $\bar{A}^N_I = -A^N_P A^P_I + A^N_I$, and $\bar{c}_I = c_I - A^P_I c_P$. Zaghrouti et al. (2014) show that if CP is infeasible or $z^{CP} \geq 0$, i.e., the objective value of CP is nonnegative, then $\bar{x}$ is an optimal solution to the SPP. Otherwise, if the columns A_j, $j \in S$ such that $S = \{j \in I : x_j > 0\}$ are pairwise row-disjoint, i.e., they do not cover the same constraints, we obtain a descent (improving) direction. S is the support of a solution to CP. S is shown to be minimal by El Hallaoui et al. (2010). S is in some sense nondecomposable using the terminology of Balas and Padberg (1975). Thus, Zaghrouti et al. (2014) show that S gives a descent direction $d_{cp} = x^* - \bar{x}$ where the lower-cost integer solution x^* is

$$
x^*_j = \begin{cases} 1, & j \in S \cup (P \setminus S^+) \text{ where } S^+ = \{k \in P : \sum_{j \in S} \bar{a}_{kj} = 1\} \\ 0, & \text{otherwise} \end{cases}
$$

S^+ is simply the index set of the leaving variables that will be replaced by variables in S. Based on this, Zaghrouti et al. (2014) propose a branching technique to eliminate the non-disjoint solutions when solving CP. They use a deep search strategy to get a descent direction.

2.3 ISUD limitations

ISUD has four main limitations:

- The computational time increases quickly with problem size: when the number of constraints increases by a factor of 2, the solution time of ISUD (mono-thread, i.e., sequential) increases by a factor of 250 or more (see Table 8).
- Since we move from an integer solution to an adjacent one at each iteration, ISUD finds one descent direction at a time. It must solve many complementary problems to reach the final solution.
- CP often produces small sets of disjoint columns, i.e., $|S|$ is small.
- ISUD uses a branching heuristic and so cannot guarantee optimality. It also lacks a measure of solution quality because it does not calculate a lower bound on the optimal solution.

Our goal is to work with small subsets of columns and to target columns with the potential to improve the objective value. We therefore solve in parallel subproblems that are likely to provide orthogonal descent directions. Moreover, parallel computing allows us to measure the solution quality without penalizing the processing time. We calculate a lower bound to measure this, and we terminate the algorithm when a specified quality is reached. These ideas are the foundation of ISU2D.

3 ISU2D approach

We now explain in more detail how our double decomposition enables us to solve the SPP more efficiently. In addition to the horizontal decomposition of ISUD, we decompose the problem vertically to find orthogonal descent directions. We use both a disjoint vertical decomposition (DVD; see Section 3.2) and an incremental vertical decomposition (IVD; see Section 3.3). In Section 3.1 we introduce ISU2D and discuss its convergence.

3.1 General structure of ISU2D

Algorithm 2 presents ISU2D; the IN-PARALLEL and END-PARALLEL terms are used to delimit multiple statements that are executed in parallel, z_{lb} and z_{ub} are the lower and upper bounds on the optimal (integer) value of SPP, and ϵ is a prespecified threshold. A brief description of ISU2D is given below; the details are in the following subsections.

Algorithm 2 ISU2D algorithm

Start with an initial integer solution $\bar{x} = x_0$, $z_{lb} = -\infty$, $z_{ub} = z_0 = c \cdot x_0$.

IN-PARALLEL

Improve $\bar{x}$ using DVD and update z_{ub}.

Improve z_{lb}. If no improvement, go to **Etq2**.

If $z_{ub} - z_{lb} \leq \epsilon$, return $x^* = \bar{x}$.

END-PARALLEL

Etq2: Improve $\bar{x}$ using IVD. Return $x^* = \bar{x}$ if $z_{ub} - z_{lb} \leq \epsilon$.

ISU2D has two main phases. In the DVD phase, it improves the current solution $\bar{x}$ using DVD. Given parameter q, DVD builds q subproblems SP_k for $k \in \{1 \dots q\}$ by partitioning P and thus T into q clusters; it solves them in parallel. We calculate a lower bound (z_{lb}) simultaneously. To get a good bound, we could solve the linear relaxation of SPP to get an LP bound and improve it by adding odd cycle cuts, clique cuts, or other facets. This capability is available in commercial solvers such as CPLEX and GUROBI. Other methods, such as Lagrangean relaxation, could be useful, particularly for large instances. Such methods give good results for a reasonable computational cost.

In the IVD phase, ISU2D sequentially solves a set of q' suproblems SP_k to optimality or near-optimality. These subproblems are based on the dual information obtained from the computation of the lower bound. In both phases we use ISUD to solve the subproblems, which are significantly smaller than the original problem. ISU2D terminates when the solution quality is satisfactory. ISU2D converges mainly because Algorithms 3 and 4 below converge. We state this more formally in the proposition below.

Proposition 1 *ISU2D is a monotonic exact algorithm that converges in a finite time.*

The proof is simple. The time of the DVD phase is at most equal to the time of computing the lower bound. We improve this bound in a finite time by solving the LP (in polynomial time if an interior point method is used) and adding a finite number of facets. The IVD phase is an improvement on ISUD, which converges in a finite time (Zaghrouti et al. 2014).

3.2 DVD phase

3.2.1 Theoretical motivation

From any integer solution to the SPP, we need at most m orthogonal descent directions (see Proposition 2), which we can obtain in parallel, to reach an optimal solution. This result motivated us to decompose the problem further to improve the performance of ISUD.

Proposition 2 *From any integer solution $\bar{x}$ to the SPP, we can reach an optimal solution x^* via at most m minimal orthogonal descent directions.*

Proof. Let $\bar{x}$ be the current solution and x^* an optimal solution. Let $D^* = \{j \in J : x^*_j - \bar{x}_j \neq 0\}$, $D^*_1 = \{j \in J : x^*_j = 1 \text{ and } \bar{x}_j = 0\}$, i.e., the index set of variables that enter the basis (optimal solution), and $D^*_0 = \{j \in J : x^*_j = 0 \text{ and } \bar{x}_j = 1\}$ be the index set of the variables that will leave the basis (current solution). We have $D^* = D^*_1 \cup D^*_0$. Obviously, the tasks covered by the entering columns are the same as those covered by the leaving columns. Therefore,

$$\sum_{j \in D^*_1} A_j = \sum_{l \in D^*_0} A_l$$

Let us now define the set sequences D^k_0, D^k_1, H^k_0, and H^k_1 as follows:

$$D^0_1 = D^*_1, D^0_0 = D^*_0, H^0_1 = \emptyset, H^0_0 = \emptyset$$

$$D^k_1 = D^{k-1}_1 \setminus H^k_1, D^k_0 = D^{k-1}_0 \setminus H^k_0, \qquad k \geq 1$$

where $H^k_1 \subset D^{k-1}_1$ is the smallest nonempty index subset of columns that could form a minimal compatible combination, and $H^k_0 \subset D^{k-1}_0$ is the index subset of columns covering the same tasks as those of H^k_1. Thus, we have

$$\sum_{j \in H^k_1} A_j = \sum_{l \in H^k_0} A_l$$

We define $k_{opt} = |\{k : H^k_1 \neq \emptyset\}|$, i.e., the number of compatible combinations formed from columns indexed by D^*_1. Clearly, $k_{opt} \leq |D^*_1|$. We have $|D^*_1| \leq |\{j = [1..n]/x^*_j = 1\}| \leq m$ (number of tasks), so $k_{opt} \leq m$.

We define the sequence of descent directions $d^k \in \mathbb{R}^n$ as $d^k_j = 1$ if $j \in H^k_1$, $d^k_j = -1$ if $j \in H^k_0$, and $d^k_j = 0$ otherwise. By construction, the d^k are orthogonal ($d^l \cdot d^h = 0$ for every $h \neq l$) and $x^* = \bar{x} + \sum_{k=1}^{k_{opt}} d^k$.

Finally, it is easy to see that the sequence x^k defined by $x^0 = \bar{x}$, $x^k = x^{k-1} + d^k$ for $k \geq 1$ is nonincreasing. Suppose there exists k_1 such that $c \cdot x^{k_1+1} > c \cdot x^{k_1}$. Then

$$c \cdot x^{k_1+1} - c \cdot x^{k_1} = c \cdot d^{k_1} > 0$$

$$c \cdot (\bar{x} + \sum_{k=1}^{k_{opt}} d^k) > c \cdot (\bar{x} + \sum_{k=1}^{k_1-1} d^k + \sum_{k=k_1+1}^{k_{opt}} d^k)$$

$$c \cdot x^* > c \cdot (\bar{x} + \sum_{k=1}^{k_1-1} d^k + \sum_{k_1+1}^{k_{opt}} d^k),$$

which contradicts the fact that x^* is optimal because $\bar{x} + \sum_{k=1}^{k_1-1} d^k + \sum_{k_1+1}^{k_{opt}} d^k$ is a feasible solution. This completes the proof. □

We note that the sequence x^k is (strictly) decreasing because $c \cdot d^k < 0$ when H^k_1 is obtained by solving the CP. Although x^* is not known at the beginning of the optimization process, we develop a method to build good approximations of the subproblems to get the sequence sets H^k_1 and then d^k in parallel; see Figure 1.

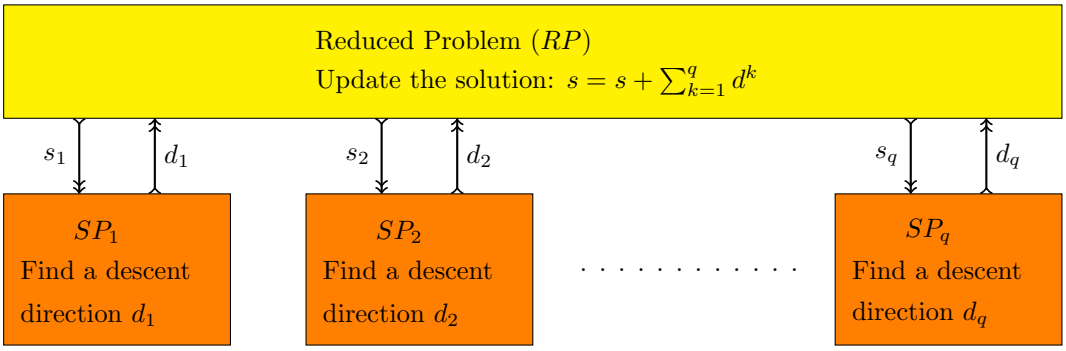

Figure 1: Decomposition and parallel descent directions

We need to minimize the impact of the overhead introduced by the parallelization. The main sources of overhead are the blocking operations and the time the processes spend communicating with each other. We must find a trade-off between computation and communication. To ensure that the benefit of parallelization outweighs the overhead, we must choose how many processes to run in parallel. We reduce the overhead by seeking descent directions (see Corollary 1) that are combinations of minimal descent directions; each subproblem finds at most one. To find these directions, ISU2D decomposes the original problem into subproblems using the splitting technique discussed in Section 3.2.2. The proof of the corollary below is omitted because it is straightforward: the result follows from orthogonality.

Corollary 1 *Any combination of minimal orthogonal descent directions is a descent direction.*

3.2.2 Splitting technique

The most novel feature of ISU2D is the use of the current solution to construct the subproblems $(SP_k)_{1\leq k\leq q}$. We partition P into q clusters where the columns indexed by cluster k cover a set of tasks T_k, i.e., $T = \cup_{1\leq k\leq q} T_k$. Let $J_k \subset J$ be the subset of columns that cover only tasks in T_k. We formulate subproblem (SP_k) as follows:

$$
\begin{aligned}
(SP_k) \qquad & min \quad \sum_{j\in J_k} c_j x_j & \\
& \sum_{j\in J_k} a_{tj} x_j = 1 & \forall t \in T_k \\
& x_j \in \{0,1\} & \forall j \in J_k
\end{aligned}
$$

Hence, building the subproblems reduces to defining the sets T_k, $1 \leq k \leq q$. We define a weighted graph $G(V, E)$ where each positive-valued variable in the current solution is represented by a vertex v of V. For simplicity, V is the index set of these variables. Let $(v, v') \in V^2$, $J_{vv'} = \{l \in J \ : \ A_v \cdot A_l \neq 0 \text{ and } A_{v'} \cdot A_l \neq 0\}$. We define E as the set $\{(v, v') : J_{vv'} \neq \emptyset\}$. Let H_0^k and H_1^k be defined as above for $k \in 1..k_{opt}$. Note that $H_0^k \subseteq V$.

Proposition 3 *The subgraph induced by H_0^k, denoted $G(H_0^k)$, is a connected component of G.*

Proof. Suppose that $G(H_0^k)$ is not a connected component of G. Note that H_1^k is minimal by construction. Let $H_0' \subsetneq H_0^k$ be such that $G(H_0')$ is the smallest connected component of $G(H_0^k)$. Then there exists $H_1' \subset H_1^k$ such that

$$\sum_{j\in H_1'} A_j = \sum_{l\in H_0'} A_l$$

Therefore, H_1' is a compatible combination by Definition 1. This contradicts the fact that H_1^k is minimal since H_1' is a strict subset of H_1^k. □

We partition G into q equally sized subgraphs $G_k = (E_k, V_k), 1 \leq k \leq q$ in a such a way that the connected components characterized by Proposition 3 are likely to be (fully) in these subgraphs. We assign a weight $w_{vv'}$ to every edge $(v, v') \in E$ (see Section 3.2.3) and partition the graph so that the total weight of the cut (i.e., the edges having their ends in different subgraphs) is minimized. The variables indexed by V_k cover a set of tasks T_k. Thus, we obtain the decomposition $\tau = \{T_k, 1 \leq k \leq q\}$, and the subproblems $(SP_k)_{1 \leq k \leq q}$ are built accordingly. Conflicting variables, i.e., those covering tasks in two different task clusters, are set to 0. The set of conflicting variables changes from iteration to iteration.

Proposition 4 *If $\bar{x}$ is not optimal, at least one conflicting variable will be positive in any improving solution.*

Proof. Let $\bar{x}^k$ be an optimal solution to SP_k. Then $(\bar{x}_{J_1}, ..., \bar{x}_{J_q})$ is an optimal solution to the SPP restricted to the variables indexed by $\cup_{k \in 1..q} J_k$, because the matrix of this restricted problem called SPP_R is block-angular. Suppose there exists an improving solution $\bar{x}'$ where all the conflicting variables are 0. Then $(\bar{x}'_{J_1}, ..., \bar{x}'_{J_q})$ is an improving solution to SPP_R, which contradicts the fact that $(\bar{x}_{J_1}, ..., \bar{x}_{J_q})$ is an optimal solution to SPP_R. □

The splitting is done in such a way that some potential conflicting variables become nonconflicting in the next iteration. Figure 2 illustrates the splitting technique on an example with 8 tasks and 10 columns. Figure 2(a) shows the graph $G(V, E)$. The cut shown as a dashed line on Figure 2(b) gives two subproblems: (SP_1) with $T_1 = \{1, 2, 3, 4, 5\}$ and (SP_2) with $T_2 = \{6, 7, 8\}$. The index set of conflicting variables is $\{5\}$.

Table 1: Example with 8 tasks and 10 columns

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
c_j	3	2	2	1	2	1	1	2	1	1
T_i/x_j	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
1	1	0	0	0	0	0	0	0	1	0
2	1	0	0	0	0	0	0	0	1	0
3	0	1	0	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0	0	0	0
5	1	0	0	0	0	1	0	0	0	0
6	0	0	0	1	0	0	1	0	0	0
7	0	0	1	0	1	0	1	0	0	0
8	0	0	1	0	0	0	0	1	0	1

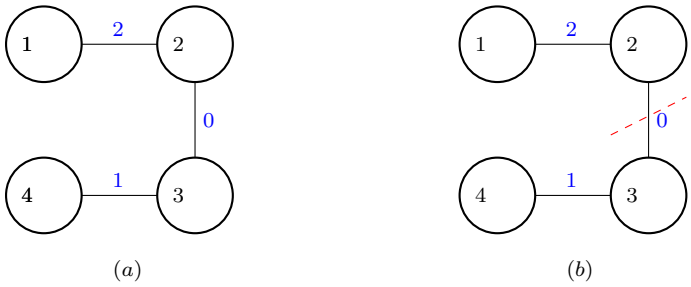

Figure 2: Example of ISU2D splitting

The weight of the edge $(v, v') \in E$ measures the likelihood that the variables indexed by $J_{vv'}$ will improve the objective value. When the edge (v, v') is not cut (e.g., edge $(1, 2)$ in Figure 2), A_v and $A_{v'}$ are grouped into a cluster and will be considered in the same subproblem. Thus, the variables indexed by v and v' could be part of a descent direction (as leaving variables). When the edge (v, v') is cut (e.g., edge (2,3) in Figure 2), it is not possible in the current iteration to improve the objective value with the variables indexed by $J_{vv'}$. Thus, a good splitting technique should avoid cutting edges (v, v') where at least one variable indexed in $J_{vv'}$ is a part of an optimal solution. Splitting depends heavily on the edge weights and consequently on the formulas used to calculate them, which we call weighting methods.

3.2.3 Weighting methods

Good weighting methods use the problem structure to decide which columns to group and which to separate. For this proof of concept, we tested some generic weighting methods and retained two promising ones:

$$w_1 : (v, v') \mapsto w_{vv'} = |\{j \in J_{vv'} \ : \ \bar{c}_j \leq 0\}|$$

$$w_2 : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\bar{c}_j : \ j \in J_{vv'}\}).$$

We use reduced costs with respect to a dual vector α derived from the current solution $\bar{x}$ such that $\bar{c}_j = c_j - \alpha \cdot A_j = 0, \forall j \in supp(\bar{x})$, i.e., the reduced costs of these basic variables are null. An infinite number of vectors α satisfy this equation. A simple one is

$$\alpha_t = \sum_{j \in J} \frac{\bar{x}_j * c_j * a_{tj}}{n_j}, \quad t \in T \tag{11}$$

where n_j is the number of tasks covered by A_j, $j \in P = supp(\bar{x})$. Equation 11 means that we associate with a task t a dual value that is the average cost per task. In the example above, $\alpha_t = 1$, $\forall t \in \{1 \dots 8\}$ and the reduced costs are given in Table 2. A more sophisticated option is $\alpha = (\alpha^{T_1}, \alpha^{T_2}, \dots, \alpha^{T_q})$ where α^k is the solution of the dual of the CP (7)–(10) when solving SP_k by ISUD.

Table 2: Example with 8 tasks and 10 columns: reduced costs

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
$\bar{c}_j$	0	0	0	0	0	-1	-1	1	-2	0

The first weighting method w_1 stipulates that a descent direction is more likely to exist in regions where there are more variables with negative reduced costs. We therefore cut the edges with the smallest number of negative reduced costs. Based on this, w_1 associates with each edge (v, v') the number of negative reduced cost columns that $J_{vv'}$ contains. The second weighting method increases the chances of getting a large step (improvement in the objective value) provided that a descent direction exists. We assume that one of the entering variables has the smallest negative reduced cost and hope to realize a large improvement in the objective value. Consequently, w_2 associates with the edge (v, v') the absolute value of the smallest negative reduced cost column indexed by $J_{vv'}$. The weights computed with this second method are illustrated in Figure 2. We hence obtain two variants of ISU2D depending on the weighting method that we use. The decomposition is adjusted dynamically.

3.2.4 DVD algorithm

Algorithm 3 outlines the DVD procedure.

Algorithm 3 DVD pseudocode

$r = 1$.

Etq1: Build τ^r and consequently SP_k, $k \in \{1 \dots q\}$ by splitting as in Section 3.2.2.

Solve in parallel the subproblems SP_k, $k \in \{1 \dots q\}$ using ISUD.

$\bar{x} = \bar{x} + \sum_{k=1}^{q} d^k$.

$r = r + 1$.

If $c \cdot (\sum_{k=1}^{q} d^k) < 0$, go to **Etq1**.

Else if $q > 2$, decrease q and go to **Etq1**.

End if.

Algorithm 3 is monotonic because $c \cdot (\sum_{k=1}^{q} d^k) \leq 0$. Actually, d^k is itself the sum of the descent directions (d_{cp}) obtained in the ISUD iterations and consequently $c \cdot d^k \leq 0$, $\forall k \in \{1 \dots q\}$. At each iteration,

Algorithm 3 explores (see Proposition 5) a different neighborhood around the current solution $\bar{x}$ by using the relative dual information. Let τ^r and $\tau^{r'}$ be two decompositions obtained at iterations r and r' respectively.

Proposition 5 *We have* $\tau^r \neq \tau^{r+1}$.

Proof. If $r' = r + 1$, there are two cases: i) The solution is not improved: in this case, we decrease q. Consequently, $\tau^r \neq \tau^{r'}$ because $|\tau^r| \neq |\tau^{r'}|$. ii) The solution is improved: q remains the same but in the splitting at least one variable x_{j_0} previously conflicting with τ^r becomes nonconflicting with $\tau^{r'}$. Let $\tau^r = \{T_k, 1 \leq k \leq q\}$ and $\tau^{r'} = \{T'_k, 1 \leq k \leq q\}$, and let T_0 be the set of tasks that are covered by x_{j_0}. Then there exists $k \neq k'$ such that $T_0 \cap T_k \neq \emptyset$ and $T_0 \cap T_{k'} \neq \emptyset$. At the same time, there exists k'' such that $T_0 \subset T'_{k''}$. Consequently, $T'_{k''}$ intersects both T_k and $T_{k'}$ but is different from them. Clearly, $\tau^r \neq \tau^{r'}$. □

Corollary 2 *We have* $\tau^r \neq \tau^{r'}$, $\forall r \neq r'$.

Proof. Suppose without loss of generality that $r \leq r'$. We consider the case $r' - r \geq 2$; the other case is discussed in Proposition 5. Suppose that $\tau^r = \tau^{r'}$. Then the solution did not change between iterations r and r' because the decomposition is the same and we assume that the subproblems are solved to optimality (in parallel). Thus, the solution is not improved between r and $r + 1$. We have a contradiction because in this case $\tau^r \neq \tau^{r'}$, as shown by Proposition 5. □

3.3 IVD phase

IVD is an improvement strategy that explores near-optimal (LP) neighborhoods. The idea is to seek an optimal or near-optimal solution by solving q' subproblems based on the variables' reduced costs with respect to the duals for an improved lower bound. IVD starts from the solution of the DVD phase and so has good upper and lower bounds (and the related dual information). IVD uses the well-known fixation technique to reduce the problem size. Here, $\bar{c}_j$ is the dual reduced cost of variable x_j computed with the LP dual values μ (so $\bar{c}_j \geq 0$ for all variables), $z_{ub} = c \cdot \bar{x}$ is the upper bound, and z_{lb} is the lower bound. If $\bar{z}_{lb} + \bar{c}_j > z_{ub}$, we remove column A_j from the constraint matrix (fixing its variable x_j to 0). This is a well-known theoretical result. This decomposition becomes very useful as we approach optimality because the gap between the upper and lower bounds is small, so we apply this to complete the solution process. Algorithm 4 outlines the procedure; q' is a parameter tuned by experimentation.

Algorithm 4 IVD pseudocode

Price columns using μ (i.e., compute their reduced costs).

Sort the variables in increasing order of reduced cost and reindex them.

For $k = 1$ to q'

For all j, if $\bar{z}_{lb} + \bar{c}_j > z_{ub}$, $J = J \setminus \{j\}$, i.e., $x_j = 0$.

Build SP_k by considering the first $k\frac{|J|}{q'}$ variables.

Solve SP_k with ISUD.

$\bar{x} = \bar{x} + d^k$, $z_{ub} = c \cdot \bar{x}$.

If $z_{ub} - z_{lb} \leq \epsilon$, terminate ISU2D.

End for.

Algorithm 4 is also monotonic because we locally and strictly improve a solution at each iteration. It is finite because the problem is bounded.

4 Computational results

In this section, we compare mono- and multi-thread versions of the standard methods, namely CPLEX and ISUD, to ISU2D with three different splitting methods: ISU2D$_1$ and ISU2D$_2$ use w_1 and w_2 respectively (see Section 3.2.3), and ISU2D$_3$ applies multistage splitting: w_2 followed by w_1. Our results show that ISU2D is faster and gives a better solution than the other methods.

We implemented ISU2D using C^{++} and the MPI (Message Passing Interface) library. The library provides parallelization and communication between processes. We use the bipartition algorithm (Kernighan and Lin 1972) to partition the weighted graph G. The tests were performed on a Unix Dell Precision T1700 with a 3.30 GHz Intel Xeon E3-1226 V3 Quad-Core processor. The computational times are in seconds.

4.1 Instances

We test ISU2D on the aircrew and bus driver scheduling instances used by Zaghrouti et al. (2014). The 90 instances require reoptimization after a perturbation because of unforeseen events. The reoptimized schedules usually have many components in common with the original schedules. In practice, there are generally penalties in the objective function to discourage changes, so many schedules are unchanged in the reoptimized solution. For a given instance, we define the perturbation ratio ρ to be the percentage of columns of the reoptimized solution that are present in the original solution. This is a good indicator of the difficulty of an instance: the larger ρ, the harder the instance.

We grouped the instances into three sets (small, medium, and large) and then into three subsets (easy, moderate, and hard) corresponding to $\rho = 50\%$, 65%, and 80% respectively. Each subset contains 10 instances. The aircrew scheduling instances are small, and we use the prefix AS. The bus driver scheduling instances are medium and large, with the prefixes BM and BL. We add ρ to the instance name to indicate the level of difficulty. We also add an incremental value to the name, e.g., BM80 indicates the set of hard medium bus driver scheduling instances, and BM80-2 indicates instance number 2 of the same set.

Table 3 lists the characteristics of the instances: the average number of rows and columns and the average density (number of nonzero elements per column). The bus driver instances have an average of 40 nonzero elements per column, which is large. This makes the problem difficult for a traditional method such as CPLEX because of the severe degeneracy and branching difficulties. The aircrew problems have low density (9 flights) and are easier to solve.

Table 3: Instance characteristics

Set	#rows	#columns	Density
Small aircrew scheduling (AS)	803	8904	9
Medium bus driver scheduling (BM)	1200	130000	40
Large bus driver scheduling (BL)	1600	570000	40

4.2 Testing methodology

We present aggregated results for the aircrew instances (Section 4.3) and then the bus driver instances (Section 4.4). The detailed results are in the Appendix. For each class of instances, we compare the three variants of ISU2D, and then we compare these variants to CPLEX (multi-thread version) and ISUD (mono- and multi-thread versions).

For each variant of ISU2D, we report information for each phase. For DVD, we report the number of iterations (#Itr), the time, and the improvement percentage (%Imp), i.e., $Imp = 100 * \frac{z_0 - z_f}{z_0 - z^*}$ where z_f is the final objective value obtained by DVD. For IVD, we report the percentage of columns (%Cols) and the time to obtain the optimal value. Note that the averages are computed by considering only the instances that ISUD solved to optimality.

We use the following ratios to compare ISU2D to CPLEX and ISUD:

- The time reduction ratio $T_a(b)$ between two algorithms a and b is defined as $T_a(b) = 100 * \frac{t(b)-t(a)}{t(b)}$ where $t(a)$ and $t(b)$ are the computational times of a and b. We write C to indicate CPLEX, I_s for the sequential (i.e., mono-thread) ISUD, and I_p for the parallel (i.e., multi-thread) ISUD. For example, T_C is the time reduction factor of the current algorithm compared to CPLEX.
- The gap between the (final) solution value $z(a)$ returned by algorithm a and the optimal value z^* is defined as $Gap(a) = 100 * \frac{z(a)-z^*}{z^*}$.

4.3 Aircrew scheduling results

Table 4 summarizes the results of the ISU2D variants on the aircrew instances; the detailed results are in Table 9. We observe that DVD has beter performance in ISU2D$_2$ than in ISU2D$_1$: %Imp is significantly higher in ISU2D$_2$ (for a similar number of iterations and execution time). The number of instances solved to optimality (#Opt) by DVD is 17 out of 30 in ISU2D$_2$ and 11 out of 30 in ISU2D$_1$. The success rate is increased by 54%. This may be because w_2 minimizes the reduced cost value (the improvement is proportional to this value) whereas w_1 maximizes the number of negative reduced cost columns regardless of the reduced cost value.

For DVD in ISU2D$_3$ the success rate is increased to 22 out of 30. The percentage of columns needed in IVD to obtain an optimal solution is significantly lower in ISU2D$_3$. The success rate is 10 out of 10 in AS50 and decreases as the difficulty increases.

Table 4: Summary of ISU2D results for small aircrew scheduling instances

	ISU2D$_1$						ISU2D$_2$						ISU2D$_3$					
	DVD				IVD		DVD				IVD		DVD				IVD	
Set	#Itr	Time	%Imp	#Opt	Time	%Cols	#Itr	Time	%Imp	#Opt	Time	%Cols	#Itr	Time	%Imp	#Opt	Time	%Cols
AS80	4	2.4	72	2	4.0	77	5	2.1	68	2	4.9	63	6	2.5	80	4	2.6	47
AS65	5	2.5	93	5	2.4	47	5	2.2	94	7	1.2	20	5	2.3	97	8	0.5	13
AS50	4	2.1	94	4	2.1	57	5	2.3	99	8	0.1	20	5	2.4	100	10	-	-

Table 5 shows the average solution times of CPLEX and ISUD on the aircrew instances and compares them to the ISU2D results. Table 10 gives the detailed results (the CPLEX gap is 0 for all these instances, and we compute the reduction factor for only the instances where the gap is less than 1%). ISU2D$_3$ has the shortest computational time. It outperforms CPLEX: it is four times faster on easy instances and at least twice as fast on hard instances. As expected, ρ influences the ISU2D results: the lower the value of ρ, the better the performance of ISU2D.

ISU2D$_3$ performs well against ISUD: ISU2D$_3$ is almost twice as fast as mono-thread ISUD and almost one and a half times faster than multi-thread ISUD. We note that multi-thread ISUD is only slightly faster than mono-thread ISUD. The time reduction is around 10% because of the size of the instances and the high overhead. This shows that the time reduction of ISU2D comes from the decomposition methods and not from the parallelization.

Table 5: ISU2D vs. CPLEX and ISUD: Aircrew scheduling instances

	C	I_s	I_p	ISU2D$_1$			ISU2D$_2$			ISU2D$_3$		
Set	Time	Time	Time	T_C	T_{I_s}	T_{I_p}	T_C	T_{I_s}	T_{I_p}	T_C	T_{I_s}	T_{I_p}
AS80	10.2	8.6	7.1	36.3	24.4	8.4	31.4	18.6	1.4	50.0	40.7	28.2
AS65	9.8	5.9	5.1	50.0	16.9	3.9	65.3	42.4	33.3	71.4	52.5	45.1
AS50	10.1	5.1	4.4	58.4	17.6	0.0	76.2	52.9	42.8	76.2	52.9	42.8

Figure 3 shows the evolution of the objective value over time for CPLEX, ISUD, and the ISU2D variants on instance AS80-1. The algorithms start from the same initial solution with an objective value equal to 376243. This behavior is typical and representative of the other instances. We connect the points in this figure to

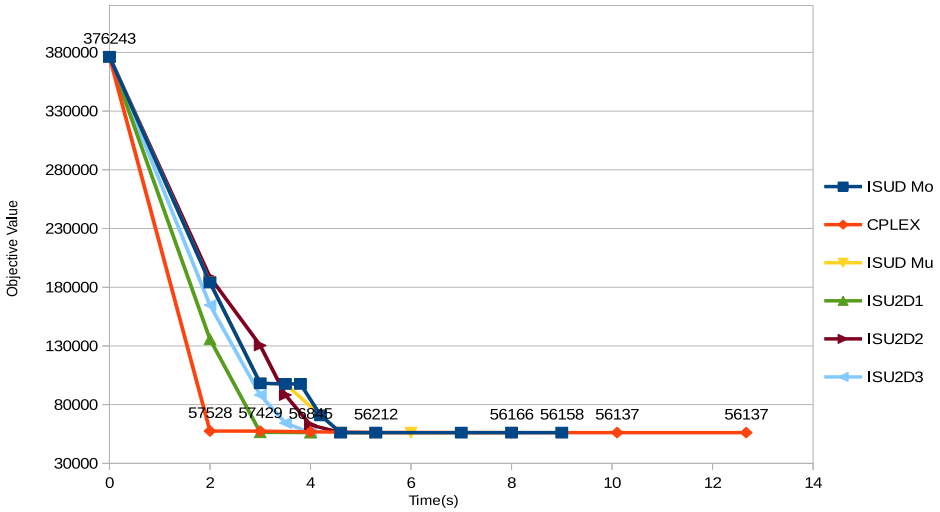

Figure 3: Evolution of objective value over time

improve its readability. The ISU2D curves decrease more sharply than the ISUD curves because ISU2D finds multiple descent directions in parallel. The ISUD curves are almost constant at the end of the solution process because ISUD branching is heuristic and we terminate only when the last branching node is infeasible. ISU2D instead uses the LP value to terminate as soon as the objective value is satisfactory. $ISU2D_3$ improves the ISUD solution time by 40% on this instance. The CPLEX curve initially decreases rapidly, but the rate slows as it approaches an optimal solution. The ISU2D variants reach an optimal solution more quickly. The three variants share similar behavior. We can rank the algorithms from best to worst (on this instance) as follows: $ISU2D_3 \gg ISU2D_2 \gg ISU2D_1 \gg$ Multi-thread ISUD $\gg$ Mono-thread ISUD $\gg$ CPLEX.

4.4 Bus driver scheduling instances

CPLEX was unable to solve any bus driver instance within a time limit of half an hour for medium instances and two hours for large instances. Note that CPLEX starts from the same initial solutions as ISUD and ISU2D. We instead focus on comparing ISU2D and ISUD. The ISU2D variants solve all the medium and large instances to optimality whereas ISUD fails to find the optimal solution for 40% of the medium and difficult instances.

Table 6 presents the same information for the bus driver instances that Table 4 presented for the aircrew instances; the detailed results are in Tables 11 and 13. The number of instances solved to optimality (#Opt) by DVD is 6 in $ISU2D_1$, 14 in $ISU2D_2$ and 22 in $ISU2D_3$. The success rate is almost quadrupled. As explained in Section 4.3, $ISU2D_3$ has a better splitting approach. Again, the percentage of columns needed to obtain an optimal solution in IVD is significantly lower for $ISU2D_3$. $ISU2D_3$ solves 46% of the large instances to optimality during DVD; this percentage decreases as the difficulty level increases.

Table 6: Summary of ISU2D results for bus driver scheduling instances

	$ISU2D_1$						$ISU2D_2$						$ISU2D_3$					
	DVD				IVD		DVD				IVD		DVD				IVD	
Set	#Itr	Time	%Imp	#Opt	Time	%Cols	#Itr	Time	%Imp	#Opt	Time	%Cols	#Itr	Time	%Imp	#Opt	Time	%Cols
MB-80	2	34	44	0	28	40	3	33	46	0	28	40	3	32	50	0	26	40
MB-65	2	33	58	0	48	41	3	33	58	0	34	42	3	34	60	1	36	38
MB-50	2	33	68	0	19	32	3	36	92	5	6	15	3	37	94	7	7	14
LB-80	3	206	41	0	244	43	3	198	41	0	189	43	4	200	44	0	164	43
LB-65	3	233	67	1	269	43	3	231	80	4	160	29	4	287	87	6	79.4	18
LB-50	3	218	88	5	139	25	3	206	88	5	108	26	3	225	95	8	38	10

Table 7 shows the average solution times of CPLEX and ISUD for the bus driver instances and compares them to the ISU2D results. Tables 12 and 14 give the detailed results. We note that the percentages and their averages are computed for only the instances that were solved to optimality using ISUD. ISU2D$_3$ is again the best variant of ISU2D. ISU2D$_3$ outperforms ISUD: on the large instances, it is four times faster than the sequential ISUD and three times faster than the multi-thread ISUD. On the medium instances, the reduction factor is lower because ISUD performs well. As the instance size increases, the reduction factor becomes more important because the size of the CP increases and consequently the solution time (e.g., for the simplex algorithm) increases much more (the observed complexity of the simplex algorithm is m^2n where m is the number of rows and n the number of columns). As expected, ρ influences the results of the ISU2D variants: the lower the value of ρ, the better the performance of ISU2D.

In contrast to the small instances, the multi-thread ISUD is twice as fast as the mono-thread ISUD because the instances are large enough and the overhead is rather small. However, ISU2D is much faster than the multi-thread ISUD. The time reduction is much more important when the decomposition method is used.

Table 7: ISU2D vs. ISUD: bus driver scheduling instances

Set	I_s time	I_p time	ISU2D$_1$ T_{I_s}	ISU2D$_1$ T_{I_p}	ISU2D$_2$ T_{I_s}	ISU2D$_2$ T_{I_p}	ISU2D$_3$ T_{I_s}	ISU2D$_3$ T_{I_p}
MB-80	295	145	66.7	31.1	68.6	36.7	68.4	36.3
MB-65	241	109	61.3	18.1	65.9	27.2	64.2	23.8
MB-50	119	60	48.2	11.0	56.6	26.3	54.0	22.5
LB-80	2313	1203	77.4	60.8	81.9	67.3	82.2	68.9
LB-65	1806	1012	70.4	52.1	75.0	59.9	76.1	61.6
LB-50	1076	822	54.7	46.2	62.4	54.3	66.7	59.2

Figures 4 and 5 show the evolution of the objective value over time on MB80-1 and LB80-1 respectively using the following algorithms: mono-thread ISUD, multi-thread ISUD, ISU2D$_1$, ISU2D$_2$, and ISU2D$_3$. ISU2D$_3$ is the fastest, yielding improvements of 60% and almost 75% over ISUD on these instances. The figures clearly show that the ISU2D variants outperform ISUD. The ISU2D curves decrease more sharply than the ISUD curves. Moreover, the ISUD curves are almost constant at the end of the solution process because there is no optimality proof, whereas the lower bound enables the ISU2D variants to stop when the required solution quality is obtained. On LB80-1, the ISU2D$_2$ curve decreases more rapidly than the ISU2D$_1$ curve, because ISU2D$_2$ searches for the steepest descent direction at each iteration. ISU2D$_3$ is better than ISU2D$_2$. We can rank the algorithms from best to worst as follows: ISU2D$_3$ $\gg$ ISU2D$_2$ $\gg$ ISU2D$_1$ $\gg$ Multi-thread ISUD $\gg$ Mono-thread ISUD.

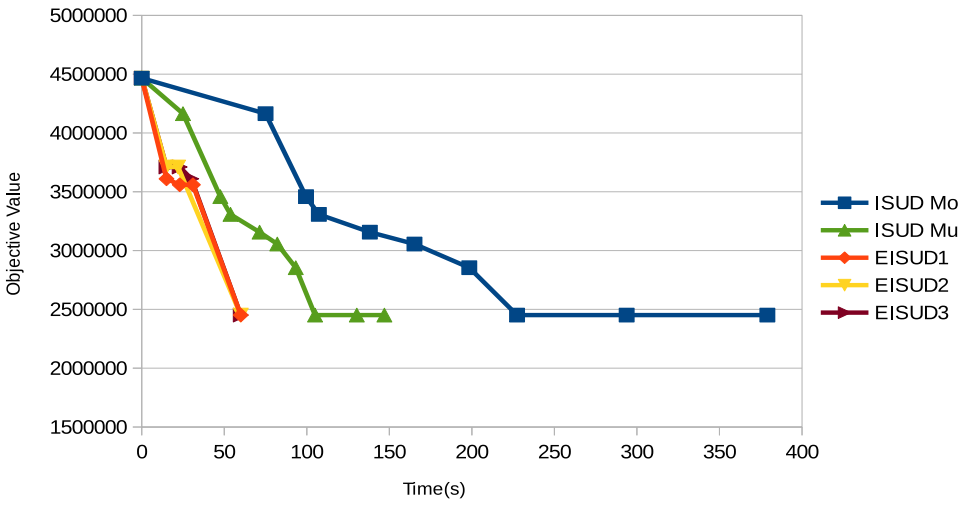

Figure 4: Evolution of mono-thread ISUD, multi-thread ISUD, ISU2D$_1$, ISU2D$_2$, and ISU2D$_3$ on MB80-1

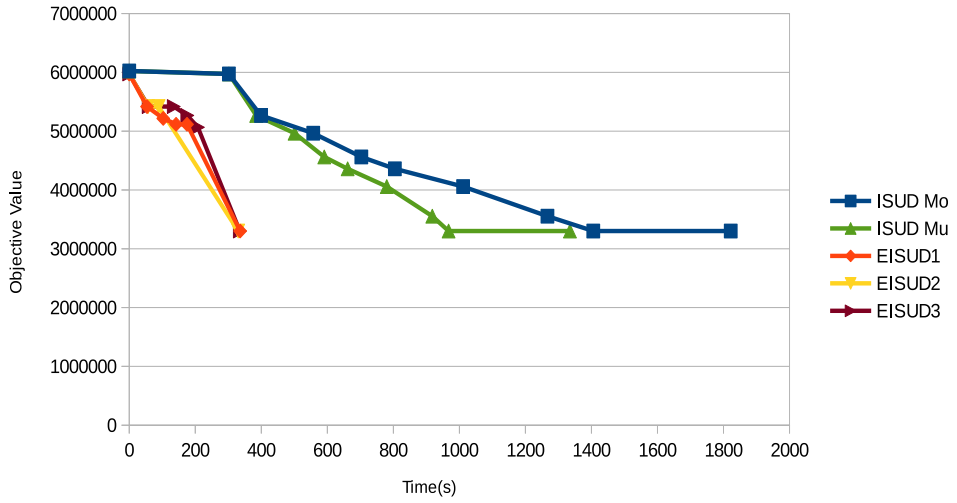

Figure 5: Evolution of mono-thread ISUD, multi-thread ISUD, ISU2D$_1$, ISU2D$_2$, and ISU2D$_3$ on LB80-1

4.5 Summary of results

Table 8 summarizes the results. ISU2D$_3$ is almost twice as fast as ISUD and almost three times faster than CPLEX on the aircrew tests. It is almost three times faster than mono-thread ISUD and almost 1.5 times faster than multi-thread ISUD on the medium bus driver tests. It is four times faster than mono-thread ISUD and three times faster than multi-thread ISUD on the large bus driver tests. We can rank the algorithms from best to worst as follows: ISU2D$_3$ $\gg$ ISU2D$_2$ $\gg$ ISU2D$_1$ $\gg$ Multi-thread ISUD $\gg$ Mono-thread ISUD.

Table 8: Summary of results

Set	Mono ISUD Time	Multi ISUD Time	CPLEX Time	ISU2D$_3$ Time	ISU2D$_3$ T_{I_s}	ISU2D$_3$ T_{I_p}
AS: Small aircrew scheduling	6.6	5.5	10.3	3.4	48.6	41.8
MB: Medium bus driver scheduling	181.3	83.8	-	56.4	62.4	27.9
LB: Large bus driver scheduling	1622.4	969.6	-	336.5	72.4	62.6

5 Conclusion

We have introduced the ISU2D double decomposition to find, in parallel, descent directions leading to improved integer solutions. Our approach is applicable to vehicle and crew scheduling SPPs. It is especially beneficial for large problems. We implemented three ISU2D variants and discussed their performance. They differ in the splitting method that they use, and we demonstrated that multistage splitting is the best. ISU2D was able to find optimal solutions for all the instances in less time than that required by ISUD and CPLEX.

Future research on splitting methods should be done to further improve the ISU2D performance. Splitting methods could be based on time-space decomposition and business rules that are specific to the application. Good dual values and and good weighting methods are the ingredients for a good ISU2D. In addition, combining ISU2D with heuristics that produce good initial solutions should significantly reduce the computational time for large SPPs.

6 Appendix: Detailed results

Table 9: ISU2D results for small aircrew scheduling instances

	ISU2D$_1$						ISU2D$_2$						ISU2D$_3$					
	DVD			IVD		Final	DVD			IVD		Final	DVD			IVD		Final
Instance	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap
AS80-1	6	4	100	-	-	0	4	2	59	66	3	0	5	2	66	67	2	0
AS80-2	3	2	44	66	3	300	7	3	100	-	-	0	7	3	100	-	-	0
AS80-3	5	2	37	100	10	0	6	2	50	100	13	0	6	2	50	100	12	0
AS80-4	4	2	80	100	1	0	4	2	58	100	6	0	5	2	72	100	2	0
AS80-5	3	2	91	100	3	0	5	2	68	66	5	3	5	2	68	66	5	3
AS80-6	7	3	100	-	-	0	4	2	78	66	6	108	11	4	100	-	-	0.3
AS80-7	2	2	88	100	10	2	3	2	77	66	3	0	7	3	100	-	-	0
AS80-8	3	2	46	100	10	291	5	2	100	-	-	0	5	2	100	-	-	0
AS80-9	3	2	64	100	3	0	4	2	35	100	9	0	5	2	58	66	5	0
AS80-10	4	2	58	100	5	0	5	2	62	66	5	170	6	2	62	66	5	100
AS65-1	7	3	100	-	-	0	5	2	100	-	-	0	5	2	100	-	-	0
AS65-2	4	3	100	-	-	0	7	4	100	-	-	0	7	4	100	-	-	0
AS65-3	4	2	98	100	2	0	6	2	91	67	1	0	6	2	91	66	1	0
AS65-4	3	2	90	100	3	0	5	2	100	-	-	0	5	2	100	-	-	0
AS65-5	3	2	74	100	9	0	5	2	100	-	-	0	5	2	100	-	-	0
AS65-6	4	2	100	-	-	0	4	2	100	-	-	0	4	2	100	-	-	0
AS65-7	3	2	71	100	7	0	4	2	81	67	4	0	4	2	81	67	4	0
AS65-8	4	2	96	67	3	0	4	2	100	-	-	0	4	2	100	-	-	0
AS65-9	10	5	100	-	-	0	3	2	66	67	7	0	6	3	100	-	-	0
AS65-10	4	2	100	-	-	0	6	2	100	-	-	0	6	2	100	-	-	0
AS50-1	4	2	94	100	2	0	5	2	96	100	1	0	6	3	100	-	-	0
AS50-2	4	2	84	100	2	0	5	2	100	-	-	0	5	2	100	-	-	0
AS50-3	4	2	100	-	-	0	4	2	100	-	-	0	4	2	100	-	-	0
AS50-4	3	2	78	67	5	4	6	2	100	-	-	0	6	2	100	-	-	0
AS50-5	4	2	100	-	-	0	5	2	100	-	-	0	5	2	100	-	-	0
AS50-6	5	2	100	-	-	0	4	2	100	-	-	0	4	2	100	-	-	0
AS50-7	4	2	96	100	2	0	4	2	100	-	-	0	4	2	100	-	-	0
AS50-8	3	2	89	100	10	1	8	4	99	-	-	1	12	5	100	-	-	0
AS50-9	5	3	100	-	-	0	4	2	100	-	-	0	4	2	100	-	-	0
AS50-10	2	2	75	100	3	0	4	2	100	-	-	0	4	2	100	-	-	0

Table 10: ISU2D vs. ISUD and CPLEX on small aircrew scheduling instances

	C	I_s		I_p		ISU2D$_1$			ISU2D$_2$			ISU2D$_3$		
Instance	Time	Time	Gap	Time	Gap	T_C	T_{I_s}	T_{I_p}	T_C	T_{I_s}	T_{I_p}	T_C	T_{I_s}	T_{I_p}
AS80-1	13	7	0	6	0	69.23	42.86	33.33	61.54	28.57	16.67	69.23	42.86	33.33
AS80-2	9	10	0	9	0	-	-	-	66.67	70.00	66.67	66.67	70.00	66.67
AS80-3	8	15	0	9	0	-50.00	20.00	-33.33	-87.50	0.00	-66.67	-75.00	6.67	-55.56
AS80-4	6	7	0	7	0	50.00	57.14	57.14	-33.33	-14.29	-14.29	33.33	42.86	42.86
AS80-5	13	9	0	7	0	61.54	44.44	28.57	-	-	-	-	-	-
AS80-6	11	7	0	7	0	72.73	57.14	57.14	-	-	-	63.64	42.86	42.86
AS80-7	10	6	0	6	0	-20.00	-100.00	-100.00	50.00	16.67	16.67	70.00	50.00	50.00
AS80-8	17	9	0	7	0	-	-	-	88.24	77.78	71.43	88.24	77.78	71.43
AS80-9	8	9	0	7	0	37.50	44.44	28.57	-37.50	-22.22	-57.14	37.50	44.44	28.57
AS80-10	7	7	200	6	200	-	-	-	-	-	-	-	-	-
AS65-1	6	6	0	5	0	50.00	50.00	40.00	66.67	66.67	60.00	66.67	66.67	60.00
AS65-2	10	7	0	5	0	70.00	57.14	40.00	60.00	42.86	20.00	60.00	42.86	20.00
AS65-3	9	5	0	4	0	55.56	20.00	0.00	66.67	40.00	25.00	66.67	40.00	25.00
AS65-4	7	5	0	5	0	71.43	60.00	60.00	71.43	60.00	60.00	71.43	60.00	60.00
AS65-5	11	5	0	4	0	18.18	-80.00	-125.00	81.82	60.00	50.00	81.82	60.00	50.00
AS65-6	9	7	0	5	0	77.78	71.43	60.00	77.78	71.43	60.00	77.78	71.43	60.00
AS65-7	23	8	0	8	0	69.57	12.50	12.50	73.91	25.00	25.00	73.91	25.00	25.00
AS65-8	9	4	0	4	0	66.67	25.00	25.00	77.78	50.00	50.00	77.78	50.00	50.00
AS65-9	7	7	0	6	0	28.57	28.57	16.67	-28.57	-28.57	-50.00	57.14	57.14	50.00
AS65-10	7	5	0	5	0	71.43	60.00	60.00	71.43	60.00	60.00	71.43	60.00	60.00
AS50-1	22	5	0	5	0	81.82	20.00	20.00	86.36	40.00	40.00	86.36	40.00	40.00
AS50-2	9	5	0	4	0	55.56	20.00	0.00	77.78	60.00	50.00	77.78	60.00	50.00
AS50-3	5	5	0	4	0	60.00	60.00	50.00	60.00	60.00	50.00	60.00	60.00	50.00
AS50-4	9	7	0	7	0	-	-	-	77.78	71.43	71.43	77.78	71.43	71.43
AS50-5	19	4	0	3	0	89.47	50.00	33.33	89.47	50.00	33.33	89.47	50.00	33.33
AS50-6	9	5	0	3	0	77.78	60.00	33.33	77.78	60.00	33.33	77.78	60.00	33.33
AS50-7	7	4	0	4	0	42.86	0.00	0.00	71.43	50.00	50.00	71.43	50.00	50.00
AS50-8	8	5	0	5	0	-50.00	-140.00	-140.00	50.00	20.00	20.00	37.50	0.00	0.00
AS50-9	8	6	0	5	0	62.50	50.00	40.00	75.00	66.67	60.00	75.00	66.67	60.00
AS50-10	6	5	0	4	0	16.67	0.00	-25.00	66.67	60.00	50.00	66.67	60.00	50.00

Table 11: ISU2D results for medium bus driver scheduling instances

	ISU2D$_1$						ISU2D$_2$						ISU2D$_3$					
	DVD			IVD		Final	DVD			IVD		Final	DVD			IVD		Final
Instance	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap
MB80-1	3	32	45	33	28	0	2	33	37	33	27	0	3	32	42	33	28	0
MB80-2	2	34	50	34	22	0	3	33	55	34	23	0	4	32	55	33	22	0
MB80-3	3	32	57	32	19	0	3	32	67	32	29	0	3	32	67	32	28	0
MB80-4	2	35	55	33	26	0	3	34	40	33	39	0	4	33	40	33	45	0
MB80-5	3	33	35	50	33	0	3	33	30	50	31	0	3	33	40	50	31	0
MB80-6	2	36	20	65	34	0	3	35	15	32	23	0	4	34	20	32	20	0
MB80-7	2	32	55	33	69	0	3	32	85	33	15	0	2	32	85	33	15	0
MB80-8	2	34	42	33	29	0	3	33	35	33	26	0	3	32	35	33	27	0
MB80-9	1	33	42	50	43	0	3	33	65	50	24	0	3	32	65	50	25	0
MB80-10	2	34	37	34	23	0	2	32	30	34	42	0	3	31	50	34	22	0
MB65-1	3	33	44	40	62	0	3	33	50	40	40	0	3	32	50	40	33	0
MB65-2	2	32	72	36	32	0	3	32	72	36	8	0	6	54	100	-	-	0
MB65-3	1	32	59	64	118	0	3	32	50	50	42	0	3	32	50	50	34	0
MB65-4	1	34	51	36	69	0	3	33	64	36	44	0	3	32	64	36	52	0
MB65-5	2	32	47	36	40	0	3	32	50	36	37	0	3	32	50	36	37	0
MB65-6	3	34	70	32	23	0	3	32	67	32	29	0	2	32	67	33	32	0
MB65-7	1	33	48	50	44	0	2	32	39	50	84	0	2	31	39	50	83	0
MB65-8	3	33	66	35	26	0	3	34	59	35	4	0	3	33	59	33	5	0
MB65-9	1	34	47	30	40	0	3	33	44	50	34	0	3	32	44	50	36	0
MB65-10	2	33	79	50	24	0	2	33	82	50	22	0	2	32	82	50	23	0
MB50-1	2	34	69	50	11	0	3	35	100	-	-	0	3	33	100	-	-	0
MB50-2	1	33	72	34	19	4	3	33	92	34	16	0	2	32	92	34	16	0
MB50-3	2	30	85	10	2	0	3	29	94	10	2	6	5	47	100	-	-	0
MB50-4	3	33	85	35	11	0	4	52	100	-	-	0	4	52	100	-	-	0
MB50-5	2	31	52	35	28	0	3	32	100	-	-	0	3	31	100	-	-	0
MB50-6	3	32	84	33	13	0	3	32	100	-	-	0	3	32	100	-	-	0
MB50-7	3	33	64	33	14	0	2	33	76	40	21	0	2	32	76	40	22	0
MB50-8	1	33	80	20	50	0	3	53	100	-	-	0	3	52	100	-	-	0
MB50-9	2	34	72	34	33	0	2	33	84	34	15	0	2	33	84	34	14	0
MB50-10	2	33	92	34	13	0	3	32	76	34	5	0	6	44	100	-	-	0

Table 12: ISU2D vs. ISUD on medium bus driver scheduling instances

	I_s		I_p		ISU2D$_1$		ISU2D$_2$		ISU2D$_3$	
Instance	Time	Gap	Time	Gap	T_{I_s}	T_{I_p}	T_{I_s}	T_{I_p}	T_{I_s}	T_{I_p}
MB80-1	379	0.0	147	0.0	84.17	59.18	84.17	59.18	84.17	59.18
MB80-2	178	0.0	93	0.0	68.54	39.78	68.54	39.78	69.66	41.94
MB80-3	139	0.0	70	0.0	63.31	27.14	56.12	12.86	56.83	14.29
MB80-4	152	0.0	86	0.0	59.87	29.07	51.97	15.12	48.68	9.30
MB80-5	145	22.6	90	22.6	-	-	-	-	-	-
MB80-6	98	32.9	56	32.9	-	-	-	-	-	-
MB80-7	371	0.0	147	0.0	72.78	31.29	87.33	68.03	87.33	68.03
MB80-8	97	41.1	62	41.1	-	-	-	-	-	-
MB80-9	156	0.0	76	0.0	51.28	0.00	63.46	25.00	63.46	25.00
MB80-10	54	63.7	44	63.7	-	-	-	-	-	-
MB65-1	214	0.0	93	0.0	55.61	-2.15	65.89	21.51	64.95	19.35
MB65-2	134	0.0	66	0.0	52.24	3.03	70.15	39.39	59.70	18.18
MB65-3	131	10.3	69	10.3	-	-	-	-	-	-
MB65-4	238	12.3	93	12.3	-	-	-	-	-	-
MB65-5	324	0.0	124	0.0	77.78	41.94	78.70	44.35	78.70	44.35
MB65-6	141	0.0	76	0.0	59.57	25.00	56.74	19.74	54.61	15.79
MB65-7	288	0.0	131	0.0	73.26	41.22	59.72	11.45	60.42	12.98
MB65-8	111	0.0	62	0.0	46.85	4.84	65.77	38.71	65.77	38.71
MB65-9	171	0.0	78	0.0	56.73	5.13	60.82	14.10	60.23	12.82
MB65-10	179	0.0	77	0.0	68.16	25.97	69.27	28.57	69.27	28.57
MB50-1	54	0.0	44	0.0	16.67	-2.27	35.19	20.45	38.89	25.00
MB50-2	94	0.0	51	0.0	44.68	-1.96	47.87	3.92	48.94	5.88
MB50-3	71	0.0	49	0.0	56.34	36.73	56.34	36.73	33.80	4.08
MB50-4	83	0.0	51	0.0	46.99	13.73	37.35	-1.96	37.35	-1.96
MB50-5	186	0.0	73	0.0	68.28	19.18	82.80	56.16	83.33	57.53
MB50-6	57	0.0	44	0.0	21.05	-2.27	43.86	27.27	43.86	27.27
MB50-7	158	0.0	77	0.0	70.25	38.96	65.82	29.87	65.82	29.87
MB50-8	155	0.0	71	0.0	46.45	-16.90	65.81	25.35	66.45	26.76
MB50-9	263	0.0	95	0.0	74.52	29.47	81.75	49.47	82.13	50.53
MB50-10	73	0.0	44	0.0	36.99	-4.55	49.32	15.91	39.73	0.00

Table 13: ISU2D results for large bus driver scheduling instances

	ISU2D$_1$						ISU2D$_2$						ISU2D$_3$					
	DVD			IVD		Final	DVD			IVD		Final	DVD			IVD		Final
Instance	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap	#Itr	Time	%Imp	%Cols	Time	%Gap
LB80-1	3	209	52	40	127	0	3	190	28	40	120	0	3	192	28	40	129	0
LB80-2	4	217	32	38	120	0	2	206	21	38	123	0	5	210	34	38	115	0
LB80-3	3	201	62	44	461	0	4	201	70	44	209	0	4	202	70	44	212	0
LB80-4	4	203	55	37	108	0	4	196	57	37	108	0	4	196	57	37	109	0
LB80-5	2	201	17	48	172	0	4	188	32	48	251	0	4	189	32	48	275	0
LB80-6	2	202	42	47	153	0	4	194	49	47	140	0	4	194	49	47	151	0
LB80-7	3	213	26	48	154	0	2	207	13	48	457	0	5	212	30	48	147	0
LB80-8	3	200	34	42	150	0	3	194	28	42	136	0	3	195	28	42	141	0
LB80-9	2	204	28	50	889	0	4	209	66	50	150	0	4	207	66	50	152	0
LB80-10	3	212	66	40	106	0	4	200	49	40	197	0	4	203	49	40	213	0
LB65-1	2	213	100	-	-	0	3	231	100	-	-	0	3	205	100	-	-	0
LB65-2	3	214	58	44	283	0	7	213	60	44	312	0	7	524	100	-	-	0
LB65-3	2	203	81	40	158	0	4	243	100	-	-	0	4	246	100	-	-	0
LB65-4	2	187	53	44	311	0	2	190	81	44	189	0	2	189	81	44	195	0
LB65-5	3	213	49	48	374	0	3	209	60	48	130	0	3	215	72	48	146	0
LB65-6	3	215	70	60	281	0	3	216	77	60	395	0	6	487	100	-	-	0
LB65-7	3	196	61	45	453	0	2	192	56	45	247	0	3	199	70	45	232	0
LB65-8	3	207	77	50	384	0	3	236	100	-	-	0	3	208	100	-	-	0
LB65-9	3	210	53	48	248	0	4	386	100	-	-	0	4	387	100	-	-	0
LB65-10	3	201	63	46	207	0	3	196	28	46	332	0	4	201	47	46	221	0
LB50-1	2	233	100	-	-	0	3	193	88	47	98	0	4	193	100	-	-	0
LB50-2	3	202	88	53	108	0	3	202	100	-	-	0	3	201	100	-	-	0
LB50-3	2	220	100	-	-	0	3	210	100	-	-	0	3	207	100	-	-	0
LB50-4	3	211	61	46	346	0	2	203	52	46	418	0	4	214	67	46	226	0
LB50-5	2	261	100	-	-	0	2	208	76	53	213	0	5	394	100	-	-	0
LB50-6	2	212	100	-	-	0	3	212	100	-	-	0	3	209	100	-	-	0
LB50-7	3	210	79	56	150	0	3	210	79	56	151	0	3	208	79	56	154	0
LB50-8	2	211	100	-	-	0	3	204	100	-	-	0	2	200	100	-	-	0
LB50-9	2	204	68	55	469	0	3	203	85	55	203	0	4	201	100	-	-	0
LB50-10	2	222	85	42	318	0	3	222	100	-	-	0	3	223	100	-	-	0

Table 14: ISU2D vs. ISUD on large bus driver scheduling instances

	I_s		I_p		ISU2D$_1$		ISU2D$_2$		ISU2D$_3$	
Instance	Time	Gap	Time	Gap	T_{I_s}	T_{I_p}	T_{I_s}	T_{I_p}	T_{I_s}	T_{I_p}
LB80-1	1822	0.0	1335	0.0	81.56	74.83	82.99	76.78	82.38	75.96
LB80-2	2594	0.0	1199	0.0	87.01	71.89	87.32	72.56	87.47	72.89
LB80-3	1485	0.0	1121	0.0	55.42	40.95	72.39	63.43	72.12	63.07
LB80-4	2411	0.0	1060	0.0	87.10	70.66	87.39	71.32	87.35	71.23
LB80-5	2867	0.0	1391	0.0	86.99	73.18	84.69	68.44	83.82	66.64
LB80-6	1419	0.0	802	0.0	74.98	55.74	76.46	58.35	75.69	56.98
LB80-7	4518	0.0	1321	0.0	91.88	72.22	85.30	49.74	92.05	72.82
LB80-8	2544	0.0	1532	0.0	86.24	77.15	87.03	78.46	86.79	78.07
LB80-9	1947	0.0	1067	0.0	43.86	-2.44	81.56	66.35	81.56	66.35
LB80-10	1522	0.0	1204	0.0	79.11	73.59	73.92	67.03	72.67	65.45
LB65-1	1375	0.0	854	0.0	84.51	75.06	83.20	72.95	83.27	73.07
LB65-2	935	0.0	666	0.0	46.84	25.38	43.85	21.17	43.96	21.32
LB65-3	769	0.0	636	0.0	53.06	43.24	68.40	61.79	68.01	61.32
LB65-4	1403	0.0	1106	0.0	64.50	54.97	72.99	65.73	72.63	65.28
LB65-5	2064	0.0	959	0.0	71.56	38.79	83.58	64.65	82.51	62.36
LB65-6	1842	0.0	1244	0.0	73.07	60.13	66.83	50.88	73.56	60.85
LB65-7	3470	0.0	1187	0.0	81.30	45.32	87.35	63.02	87.58	63.69
LB65-8	1657	0.0	1189	0.0	67.35	54.50	85.76	80.15	85.76	80.15
LB65-9	2579	0.0	1213	0.0	82.24	62.24	85.03	68.18	84.99	68.10
LB65-10	1965	0.0	1063	0.0	79.24	61.62	73.13	50.33	78.52	60.30
LB50-1	833	0.0	602	0.0	72.03	61.30	65.07	51.66	65.43	52.16
LB50-2	852	0.0	663	0.0	63.62	53.24	76.29	69.53	76.41	69.68
LB50-3	367	0.0	359	0.0	40.05	38.72	42.78	41.50	43.60	42.34
LB50-4	712	10.7	1970	10.7	-	-	-	-	-	-
LB50-5	1215	0.0	951	0.0	78.52	72.56	65.35	55.73	67.57	58.57
LB50-6	350	0.0	329	0.0	39.43	35.56	40.29	36.47	40.29	36.47
LB50-7	3932	0.0	1040	0.0	90.84	65.38	90.82	65.29	90.79	65.19
LB50-8	649	0.0	586	0.0	67.49	63.99	68.72	65.36	69.18	65.87
LB50-9	577	0.0	520	0.0	-16.64	-29.42	29.64	21.92	65.16	61.35
LB50-10	1269	0.0	1198	0.0	57.45	54.92	82.66	81.64	82.43	81.39

References

E. Abbink, J. van't Wout, and D. Huisman. Solving large scale crew scheduling problems by using iterative partitioning. In Proceedings of the Seventh Workshop on Algorithmic Approaches for Transportation Modeling. Optimization and Systems Vol. 7, pages 96–106, 2007.

P. Alefragis, P. Sanders, T. Takkula, and D. Wedelin. Parallel integer optimization for crew scheduling. Annals of Operations Research, 1:141–166, 1999.

E. Balas and M. W. Padberg. On the set-covering problem: II An algorithm for set partitioning. Operations Research, 23:74–90, 1975.

M. L. Balinski and R. E. Quandt. On an integer program for a delivery problem. Operations Research, 12:300–304, 1964.

H. D. Chu, E. Gelman, and E. L. Johnson. Solving large scale crew scheduling problems. In Interfaces in Computer Science and Operations Research, pages 183–194. Springer, 1997.

G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. Daily aircraft routing and scheduling. Technical report, GERAD, Montreal, Canada, 1994. Research report G-94-21.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. Solomon, and F. Soumis. Crew pairing at Air France. European Journal of Operational Research, 2:245–259, 1997.

I. El Hallaoui, A. Metrane, F. Soumis, and G. Desaulniers. An improved primal simplex algorithm for degenerate linear programs. INFORMS Journal on Computing, 2010. doi: 10.1287/ijoc.1100.0425.

M. Eso. Parallel Branch and Cut for Set Partitioning. PhD thesis, Cornell University, 1999.

M. R. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., San Francisco, USA, 1979.

U. U. Haus, M. Koppe, and R. Weismanter. The integral basis method for integer programming. Mathematical Methods of Operations Research, 53(3):353–361, 2001.

K. L. Hoffman and M. Padberg. Solving airline crew-scheduling problems by branch-and-cut. Management Science, 39(6):657–682, 1993.

S. Jütte and U. W. Thonemann. Divide and price: A decomposition algorithm for solving large railway crew scheduling problems. European Journal of Operational Research, 219:214–223, 2012.

B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 49:291–307, 1972.

D. Klabjan, E. Johnson, G. Nemhauser, E. Gelman, and S. Ramaswamy. Solving large airline crew scheduling problems: Random pairing generation and strong branching. Computational Optimization and Applications, 20: 73–91, 2001.

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

J. T. Linderoth, E. K. Lee, and M. W. P. Savelsbergh. A parallel, linear programming-based heuristic for large-scale set partitioning problems. INFORMS Journal on Computing, 13(3):191–209, 2001.

M. R. Rao. Cluster analysis and mathematical programming. Journal of the American Statistical Association, 66.335: 622–626, 1971.

C. Ribeiro and F. Soumis. A column generation approach to the multiple depot vehicle scheduling problem. Operations Research, 42:41–52, 1991.

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

A. Saxena. Set-partitioning via integral simplex method. PhD thesis, Carnegie Mellon University, 2003.

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

H. Topaloglu and W. Powell. A distributed decision-making structure for dynamic resource allocation using nonlinear functional approximations. Operations Research, 53:281–297, 2005.

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