Les Cahiers du GERAD ISSN: 0711–2440

Distributed integral column generation

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

G–2018–98

November 2018

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2018
– Bibliothèque et Archives Canada, 2018

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2018
– Library and Archives Canada, 2018

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

Distributed integral column generation

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

[a] *GERAD, 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

November 2018
Les Cahiers du GERAD
G–2018–98

Abstract: The Integral Simplex Using Decomposition (ISUD) algorithm has been developed recently to solve large set partitioning problems (SPPs) in a primal way, i.e., moving from an integer solution to an improved adjacent one until optimality is reached. More recent works intended to enlarge its applications and to increase its performances. We cite namely the distribution version of ISUD called DISUD which implements the multi-agent system approach. In this work, we develop a distributed integral column generation (DICG) algorithm that extends DISUD to the column generation context in order to solve practical vehicle and crew scheduling problems. The computational tests on large bus drivers scheduling and aircrew pairing problems show that DICG gets good results and outperforms a distributed version of the well-known restricted master heuristic (RMH). DICG yields optimal or near optimal solutions in less than one hour.

Keywords: Set partitioning problems, Integral Simplex Using Decomposition, multi-agent systems, column generation

1 Introduction

Column generation (CG) is closely connected to Dantzig–Wolfe decomposition which is introduced by Dantzig and Wolfe (1960). It is widely used to solve industrial optimization problems. It involves reformulating the problem as a restricted master problem RMP and one or more column generation subproblems CGSPs. The RMP has as few variables as possible. New variables are added to the RMP as long as the solution process continues. At each iteration, the RMP is solved to get a pair of primal and dual solutions. Then, duals are used in the CG subproblem to determine if there are any columns that can improve the RMP current solution as shown on Figure 1. The algorithm stops when no negative reduced-cost columns are generated and consequently the RMP solution is also optimal for the linear relaxation of the original problem. In integer optimization problems, column generation is usually embedded in a branch and bound procedure to get an integer solution. The resulting method is known as Branch and Price (see Desrochers and Soumis 1989, Barnhart et al. 1998, Desaulniers et al. 1997, Gamache et al. 1999).

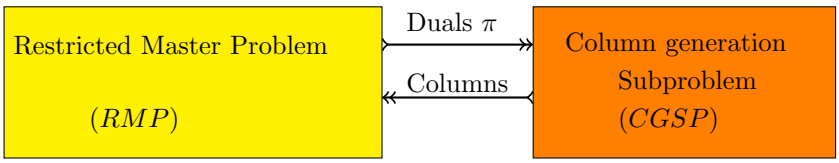

Figure 1: Column generation process

In vehicle routing and crew scheduling optimization problems, the RMP is often the well-known set partitioning problem SPP. The latter can be defined using the following scheduling terminology. A set partitioning constraint is associated with a *task* (for example, a flight leg or a bus trip to be accomplished by a *pilot or a bus driver*). Let $T = \{1, 2, ..., m\}$ be the set of tasks and $J = \{1, 2, ..., n\}$ the set of feasible schedules. With each schedule, we associate a variable x_j, a cost c_j and a column $Aj = (a_{tj})_{t \in T}$ where a_{tj} takes value 1 if Aj covers task t and 0 otherwise. The matrix $A = [A_1, A_2, ..., A_n]$ is a binary matrix. Then, the set partitioning problem formulation is:

$$\text{minimize} \quad \sum_{j \in J} c_j x_j \tag{1}$$

$$\text{(SPP) subject to} \quad \sum_{j \in J} a_{tj} x_j = 1, \qquad \forall t \in T \tag{2}$$

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

The objective function (1) seeks to minimize the total cost. The set partitioning constraints (2) ensure that each task is covered exactly once. Constraints (3) impose integrality on the x_j variables. SPP is NP-hard (see Garey and Johnson 1979). There are many heuristic and exact algorithms devoted to solve it. The most used method is the famous branch and cut (Hoffman and Padberg 1993, Desaulniers et al. 1997). However, this method becomes inefficient and takes a huge time to reach an optimal solution for large instances due to degeneracy and the size of the branching tree.

Recently, Zaghrouti et al. (2014) proposed the Integral Simplex Using Decomposition algorithm (ISUD), which is based on the Improved Primal Simplex algorithm (IPS) (see El Hallaoui et al. 2011), to solve SPP. At each iteration ISUD decomposes the original problem into two subproblems. The first, called the reduced problem (RP), which only considers columns that are *compatible with the current solution, i.e., columns that belong to the vector subspace generated by the columns associated with the current solution positive variables. The second, called the Complementary Problem (CP), contains only the columns that are incompatible with the current solution. Its main role is to find a descent direction to improve the current solution of RP. ISUD stops when neither the reduced problem nor the complementary problem can improve the current solution.* Their results show that ISUD deals more efficiently with degeneracy and is able to solve large problems that are up to 570000 variables and 1600 constraints.

Since then, several improvements have been added to the initial version of ISUD namely those of Rosat et al. (2016, 2017a) and Zaghrouti et al. (2018). Rosat et al. (2016) studied the cone of the CP directions and

proposed different formulas for the normalization constraint in order to favor the integrality of the descent direction found by the CP. In addition, Rosat et al. (2017a) proposed to add cuts to ISUD and concluded that this technique is costly in computing time especially for large instances. In the meanwhile, Zaghrouti et al. (2018) developed the Zoom algorithm which explores a neighborhood of the fractional solution, when it is not possible to find an improving "integer" direction, rather than exploring a branching tree as it is the case in the ISUD first version, see Section 2 for more details.
As the current trend in computer science is to produce multicore processors and to design parallel algorithms, Foutlane et al. (2017) developed the Integral Simplex Using Double Decomposition algorithm (ISU2D). It is a parallel algorithm based on ISUD. At each iteration, ISU2D groups the columns of the current solution into clusters in order to decompose the CP into independent complementary subproblems using the notion of compatibility defined just above. The set of complementary subproblems are solved in parallel to improve the current solution by combining the returned descent directions. ISU2D reduces the computing time of ISUD by a factor of 3 to 4 for the tested instances. ISU2D is then generalized and improved in Foutlane et al. 2018. They use the multi-agent system approach (MAS) to introduce a general framework for a distributed version of ISU2D called DISUD. It enables to use different decompositions simultaneously. Tests of DISUD on aircrew scheduling problems show that DISUD is better than DCPLEX, the distributed version of the state of the art commercial solver CPLEX. DISUD achieves better quality solutions than DCPLEX and reduces the computing time by an average factor of 4 to 5 for test instances.

In this paper, we introduce a distributed integral column generation (DICG) that combines column generation and DISUD to solve very large scale practical SPP instances. We summarize below the most important contributions of the paper:

- DICG introduces a flexible framework using a multi-agent system to parallelize the integral column generation approach where, at each iteration, we improve the current integer solution until satisfaction. Each agent finds multiple descent directions in parallel and zooms around these directions to improve the current solution more significantly. This introduces a new level in the conventional column generation method.
- We compare two versions, one competitive and another cooperative, where agents compete or cooperate. In both, they exploit the information gathered during the solution process to improve the current integer solution.
- Tests on bus crew scheduling and real crew pairing instances from the transport industry, with up to 2000 tasks (bus trips, flights) and millions of variables, show the effectiveness of DICG. We succeed to compute excellent quality solutions (gap less than 1%) for all instances.

The remainder of this paper is organized as follows. Section 2 presents briefly some useful notions on the decomposition basics and the main parts of ISUD versions. Section 3 describes the new algorithm DICG and provides a detailed algorithmic and analysis of its components. In Section 4, we discuss computational results and the effectiveness of our algorithm. Finally, we end this paper with some concluding remarks and suggestions for future research in Section 5.

2 Preliminaries

In this section, we provide the basic notions of ISUD in addition to its main improved variants. These improvements are used in DICG to solve very large SPPs more efficiently.

2.1 Decomposition basics

Given an integer solution $\bar{x}$ to SPP, let P_{int} be the index set of its positive components, i.e., $P_{int} = supp(\bar{x}) = \{j \in J : \bar{x}_j = 1\}$.

SPP could be decomposed into a reduced problem RP and a complementary problem using the following definition of compatibility (El Hallaoui et al. 2011):

Definition 1 *Given P an index set of some linearly independent columns containing at least P_{int}, a subset S of J is said to be compatible with P, 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 combination of columns, possibly a singleton, $\sum_{j\in S} v_j A_j$ is also said to be compatible. S is said to be minimal if any strict subset of it is incompatible.*

We note that the incompatibility degree of a column A_j towards a given integer solution is a measure that represents a distance of A_j from the current solution. Let C and I be the index sets of the compatible and incompatible columns respectively. They form a partition of J, i.e., $J = C \cup I, C \cap I = \emptyset$. The restriction of SPP to compatible columns only defines the reduced problem (RP) as follows :

$$\text{(RP)} \qquad \begin{aligned} &\text{minimize } z^{RP} = c_C \cdot x_C && (4)\\ &\text{subject to } A_C x_C = e && (5)\\ &x_C \in \{0,1\}^{|C|} && (6) \end{aligned}$$

When $P = P_{int}$, a pivot on any compatible column with a negative reduced cost leads to an improved integer solution according to Zaghrouti et al. (2014). Moreover, if x_C^* is an optimal solution to RP, $\bar{x} = (x_C^*, 0)$ will be a solution to SPP.

Similarly, we define the Complementary Problem (CP) as follows :

$$\text{(CP)} \qquad \begin{aligned} &\text{minimize } z^{CP} = \sum_{j\in I} c_j v_j - \sum_{l\in P} c_l \lambda_l && (7)\\ &\text{subject to } \sum_{j\in I} A_j v_j - \sum_{l\in P} A_l \lambda_l = 0 && (8)\\ &e \cdot v_I = 1 && (9)\\ &v_j \geq 0, \qquad j \in I && (10) \end{aligned}$$

In fact, the goal of the CP is to find a subset of incompatible columns to replace a subset of the current solution columns, i.e., from $supp(\bar{x})$. More precisely, we look for a convex combination of incompatible columns that is compatible and has a negative reduced cost.

Zaghrouti et al. (2014) show that $\bar{x}$ is an optimal solution to SPP when the CP is infeasible or $z^{CP} \geq 0$, i.e., the objective value of the CP is nonnegative. In the other case, the CP returns a descent direction $d = (v, -\lambda, 0)$. In this case, let $S^+ = \{j \in I : v_j > 0\}$ and $S^- = \{l \in P, \lambda_l > 0\}$ be the sets of entering and leaving variables respectively. When the columns A_j, $j \in S^+$ are pairwise row-disjoint, i.e., they do not cover the same constraints, and $S^- \subset P_{int}$, we get an *integer* descent direction leading to an improved integer solution. Moreover, S^+ is shown to be minimal by El Hallaoui et al. (2011), i.e., non-decomposable using the terminology of Balas and Padberg (1975). This means that pivoting on variables indexed by S^+ leads to an adjacent extreme integer point with better cost value. The direction d is said to be fractional when columns A_j, $j \in S^+$ are not pairwise row-disjoint. In this case, Zaghrouti et al. (2014) proposed a branching scheme to eliminate the non-disjoint solutions when solving the CP using a diving branching strategy to get an integer descent direction.

In short, ISUD is a two-stage algorithm: at each iteration, it looks first for an improved integer solution by using the RP and second by solving the CP. The algorithm stops when both cannot improve the current solution. We discuss the improvements that have been made to ISUD in the next subsection.

2.2 ISUD improvements and versions

ISUD has been the subject of intensive research to improve it. Rosat et al. (2016) replaced the normalization constraint (9) by the constraint $w \cdot v_I = 1$ and studied the influence of the weight vector w on the integrality of the descent directions returned by the CP. It is obvious that when $w = e$, we obtain the classical normalization constraint (9). They proposed new formulas to compute the weight w_j based on the number of tasks covered by the column A_j and the degree of its incompatibility.

Zaghrouti et al. (2018) have proposed a "Zoom" version to avoid implementing a complex and exhaustive branching when the complementary problem returns a fractional descent direction. They proposed to zoom around this "fractional" direction instead of branching in the classical CP. Indeed, a set of columns compatible with the fractional direction is solved by the reduced problem to get an improved integer solution.

The main steps of Zoom, as reported in Foutlane et al. (2018), are provided below :

Step 1: *Find a good heuristic initial solution* x_0 *and set* $\bar{x} = x_0$, $P = P_{int}$, $d = 0$.

Step 2: *Find a better integer solution around d:*

- *Increase* P *: set* $P = P \cup \{j : d_j > 0\}$.
- *Construct and solve RP.*
- *Update* $\bar{x}$ *and* P*: if* $\bar{x}$ *is improved, set* $P = P_{int}$.

Step 3: *Get a descent direction d:*

- *Solve the CP to get a descent direction d.*
- *If no descent direction can be found or* $|z^{CP}|$ *is small enough then stop: the current solution is optimal or near optimal.*
- *Otherwise, go to Step 2.*

Thus, when the direction is fractional, they construct RP around this direction as explained above and solve it by a MIP solver.
In addition, there are other equivalent formulations of the complementary problem. Let $A_P = \begin{pmatrix} A_P^1 \\ A_P^2 \end{pmatrix}$ be a submatrix of A composed of columns indexed by P where A_P^1 is without loss of generality composed of the first —P— linearly independent rows. .

Similarly, let $A_I = \begin{pmatrix} A_I^1 \\ A_I^2 \end{pmatrix} = (a_{ij})_{\substack{1 \le i \le m \\ j \in I}}$ be a submatrix of A composed of incompatible columns indexed by I with A_I^1 a $|P| \times |I|$ matrix.

We thus obtain an equivalent model involving only incompatible variables. In fact, constraint (8) could be written as:

$$\begin{pmatrix} A_I^1 \\ A_I^2 \end{pmatrix} v = \begin{pmatrix} A_P^1 \\ A_P^2 \end{pmatrix} \lambda$$

Observe that A_P^1 is invertible, so $\lambda = (A_P^1)^{-1} A_I^1 v$ and consequently the variables λ could be replaced. This results in the following CP formulation:

$$z_P^{CP} = \min_v \left(c_I^\top - c_P^\top \left(A_P^1\right)^{-1} A_I^1 \right) v \tag{11}$$

$$\text{(CP) subject to } \left(A_P^2 \left(A_P^1\right)^{-1} A_I^1 - A_I^2 \right) v = \mathbf{0} \tag{12}$$

$$w \cdot v = 1 \tag{13}$$

$$v \ge \mathbf{0}. \tag{14}$$

Consequently, we can use the matrix $M = (A_P^2 (A_P^1)^{-1}, -I_{|P|})$ to measure the incompatibility of A_j column by $\|MA_j\|_1$. We mention that $I_{|P|}$ is the $|P| \times |P|$ identity matrix. This measure is equal to 0 for compatible columns and positive for incompatible ones. The constraint (12) can be rewritten simply as: $MA_I v = 0$.

Foutlane et al. (2017) presented the principle of dynamic decomposition. They proposed ISU2D which finds in parallel orthogonal descent directions leading to an integer solution with a larger improvement. The approach splits the complementary problem into a set of subproblems CSP_k, defined below, where $I_k \subset I$ and $\bar{c} = \left(c_I^\top - c_P^\top \left(A_P^1\right)^{-1} A_I^1 \right)$. These subproblems are then solved in parallel.

$$\min_{v} \quad \sum_{j \in I_k} \bar{c}_j v_j \tag{15}$$

$$CSP_k \qquad MA_{I_k} v_k = 0 \tag{16}$$

$$\sum_{j \in I_k} w_j v_j = 1 \tag{17}$$

$$v_j \in \{0,1\} \qquad \forall j \in I_k \tag{18}$$

To do so, they construct a graph G(V,E), where the columns of the current solution are represented by the vertices. Then, they use a scoring function to calculate the weight $w(v_1,v_2)$ for each edge $(v_1,v_2) \in$ E based on columns and problem information. They obtain a partition of the vertices of G into some clusters that minimize the cut; in other words, a partition of P into a certain number of subsets P_k: $P = \cup P_k$, $P_k \cap P'_k =$ if $k \neq k'$ where P_k corresponds to the k^{th} cluster. CSP_k can actually be obtained from the CP by replacing P by P_k in (11)–(14); see Foutlane et al. (2017) for more details.

Foutlane et al. (2018) generalized ISU2D and proposed the DISUD, a distributed version of ISUD using a multi-agent system approach. They consider a network of worker agents where each agent dynamically splits the original CP using its own scoring function. So, the agent i constructs hence an RP and q complementary subproblems $(CSP_k^{[i]})_{1 \leq k \leq q}$.

3 DICG Algorithm

DICG is a multi-agent algorithm where a master agent coordinates a set of worker agents. Using the set of worker agents, DICG realizes multiple column generations and solves the obtained restricted master problems in parallel to get an improved integer solution. We present the worker and master agents in Sections 3.1 and 3.2 respectively.

We implemented DICG as an asynchronous algorithm in such a way that each worker does not have to wait for other agents to end their iteration to start a new iteration. Worker agents exploit the available time to improve the current solution. DICG is designed to run on more than a single machine, thus making it possible to solve large problems. DICG stops when all the agents are idle or if it reaches a limit set by the user. Such limits include a time limit, a limit on the number of iterations, a limit on the number of solutions found, or other similar criteria.

3.1 Worker agents

Worker agents realize simultaneously multiple column generations and decompositions to get an improved solution. Each worker agent starts with a warm up phase where it generates, for a certain time, a set of columns with GEN procedure using the duals sent by master agent. Each worker agent generates columns with its own parameter setting, possibly different from the parameter settings of other workers, for a limited period of time or a limited number of iterations.This phase is intended to ensure that the worker has a sufficient number of columns to start the solution process. After this phase, a worker agent runs DISUD using DVD or IVD on the generated columns and generates new ones as needed using GEN until the master agent asserts that a good quality solution is found. GEN, DVD and IVD procedures are briefly described in Sections 3.1.1, 3.1.2, and 3.1.3 respectively. More details on DVD and IVD decompositions are in Foutlane et al. (2017). An illustration of a worker agent state mode transition is given in Figure 2. The nontrivial modes are described in the subsequent subsections.

A worker agent behavior depends on the message received from the master as indicated in Algorithm 1 below.

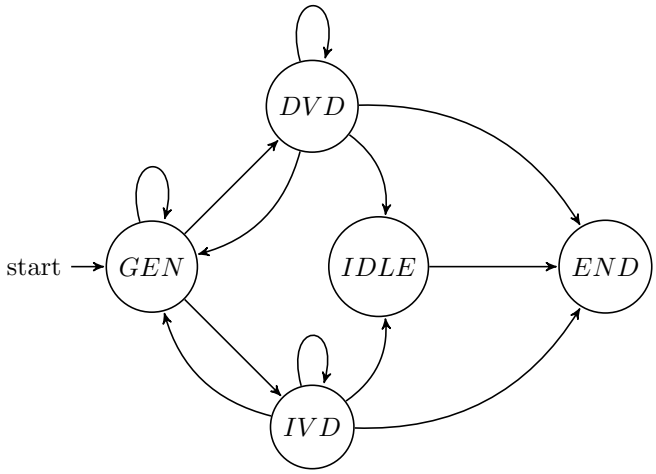

Figure 2: Worker agent basic state evolution

Algorithm 1 Worker agent algorithm

Do
  Wait for a message from the master. In case of:
    msgGEN: Set $\bar{\pi}^{[i]} = \pi_b$, generate new columns using GEN (see Section 3.1.1);
    msgSOL: Set $\bar{x}^{[i]} = x_b$
    msgMODE-DVD: Call DVD algorihm (Algorithm 2);
    msgMODE-IVD: Call IVD algorihm (Algorithm 3);
    msgMODE-IDLE: Wait;
    msgSTOP: Stop (do memory cleaning);
While (true)

We mention that throughout this paper, we use superscript [i] to denote quantities belonging to the i^{th} agent.

3.1.1 GEN mode

In this mode, a worker agent generates potential columns (feasible schedules) with reduced cost sufficiently negative. The GEN procedure consists in solving the CGSP, that is actually the shortest path problem with resource constraints ($SPPRC$), using a label-setting algorithm (see Desaulniers et al. (2005)). It is basically a dynamic programming approach (generalized Dijkstra algorithm) where dominated labels are eliminated and nodes are sorted in a topological order (the networks of the CGSP used in this paper are acyclic). To generate columns, we use the dual vector π_b sent by the master agent to price out arcs of the CG networks. During the warm up phase, the dual vector is the one that the master agent finds after the solution of the linear relaxation of the restricted master problem at a given column generation iteration (see Section 3.2). The worker agents accumulate hence a certain number of columns to start up with in the next DVD phase. In the course of the DVD phase, π_b is exactly the one obtained by concatenating dual subvectors π_k returned by the CSP_k. After that, it is the CSP dual vector π in IVD phase. We note that π_b verifies $\bar{c}_j = c_j - \pi_b \cdot A_j = 0, \forall j \in supp(\bar{x})$, i.e., the reduced costs of these basic variables are null.

3.1.2 DVD mode

During the DVD mode, the i^{th} worker agent partitions $P_{int}^{[i]} = supp(\bar{x}^{[i]})$ into q clusters, where the columns belonging to the cluster k cover a set of tasks T_k. Consequently, we have $T = \cup_{1 \leq k \leq q} T_k$ and decomposing the problem reduces to defining the partition $\tau = (T_k)_{1 \leq k \leq q}$. For the partitioning purpose, each agent constructs a weighted graph $G(V, E)$ where each column A_v, $v \in \text{supp}(\bar{x}^{[i]})$ is represented by a vertex $v \in V$. Let $(v, v') \in V^2$, $I_{vv'} = \{l \in I \ : \ A_v \cdot A_l \neq 0 \text{ and } A_{v'} \cdot A_l \neq 0\}$ and $T_{vv'}$ is the set of all tasks covered by either A_v or $A_{v'}$. The weight of edge (v, v') measures the probability that some of the variables indexed by $I_{vv'}$ could

improve the objective value if entered into the basis. The i^{th} agent uses a weighting method $we^{[i]}$ to score each edge $(v, v') \in E$. Then, it partitions the graph into q disjoint subgraphs using a min-cut algorithm (see Kernighan and Lin 1972). Finally, using its weight normalization vector $w^{[i]}$, the agent i constructs an RP and q complementary subproblems $(CSP_k^{[i]})_{1\leq k\leq q}$ formulated as follows:

$$
\begin{aligned}
& & \min \quad & \sum_{j\in I_k^{[i]}} \bar{c}_j v_j & & \\
CSP_k^{[i]} & & MA_{I_k^{[i]}} v_k &= 0 & & (19)\\
& & \sum_{j\in I_k^{[i]}} w_j^{[i]} v_j &= 1 & & (20)\\
& & v_j &\in \{0,1\} & \forall j \in I_k^{[i]} \quad & (21)
\end{aligned}
$$

where $I_k^{[i]} \subset I$ is the subset of incompatible columns that cover only tasks in T_k. We have $I^{[i]} = \cup_{1\leq k\leq q} I_k^{[i]}$. Hence, we generalize the DVD concept mentioned in Foutlane et al. (2018) by introducing the weight normalization vector while defining the $(CSP_k^{[i]})_{1\leq k\leq q}$. Consequently, the i^{th}agent behavior is defined by the pair $(w^{[i]}, we^{[i]})$ as those parameters are used to construct the partition $(CSP_k^{[i]})_{1\leq k\leq q}$ and consequently the resulting pair $(\bar{x}^{[i]}, \pi^{[i]})$ after the resolution. As in Foutlane et al. (2018), we use the following weighting methods which propose to use the reduced cost $\bar{c}_j$, the number of tasks n_j and the incompatibility degree k_j of a column A_j:

- $we_1 : (v, v') \mapsto |\{j \in I_{vv'} \ : \ \bar{c}_j \leq 0\}|$ which scores each edge (v, v') with the number of negative reduced cost columns that $I_{vv'}$ contains. This means that it is likely to find a descent direction where there are more negative reduced cost columns.
- $we_2 : (v, v') \mapsto -\min(0, \min\{\bar{c}_j \ : \ j \in I_{vv'}\})$ that associates with the edge (v, v') the absolute value of the smallest negative reduced cost column from those indexed by $I_{vv'}$. It stipulates that a descent direction contains the least negative reduced cost column.
- $we_3 : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\frac{\bar{c}_j}{n_j} \ : \ j \in I_{vv'}\})$ which takes into account the number of tasks of columns and stipulates that a good entering variable should have the smallest average negative reduced cost per task.
- $we_4 : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\frac{\bar{c}_j}{k_j} \ : \ j \in I_{vv'}\})$ that scores the edge (v, v') with the reduced cost per incompatibility degree $(k_j = \|MA_j\|_1)$ ratio. It is likely that small ratios would favor integrality of the descent direction.

In addition, we use the following weights vectors w_1 , w_2 and w_3 in the normalization constraint as defined in Rosat et al. (2016):

- $w_{1j} = 1$ which was used in the first versions of ISUD and Zoom.
- $w_{2j} = k_j$, the incompatibility degree of column A_j. This favors the direction with columns having small incompatibility degree.
- $w_{3j} = n_j$, the number of tasks covered by A_j. This favors the direction with columns covering fewer tasks.

Worker agents take benefit from ISUD improvements to get better solutions $\bar{x}^{[i]}$. Each of them explores a different region because it uses a different decomposition and normalization constraint. These latter impact the dual solution that in its turn impacts the columns generation process. Indeed, while the classical Zoom algorithm zooms around one direction using the unit weight normalization vector, DICG looks for multiple descent directions around a multitude of orthogonal directions and using different weight normalization vectors. DICG can be hence interpreted as a multi-zooming algorithm. Algorithm 2 outlines the DVD procedure of a worker agent. An illustration is shown in Figure 3. Of course, the list of agents is not exhaustive and other agents could be added easily using this framework.

Algorithm 2 DVD pseudocode for agent i

Build $\tau^{[i]}$ and consequently $CSP_k^{[i]}$, $k \in \{1 \dots q\}$ using $we^{[i]}$ and $w^{[i]}$.
Solve in parallel the $CSP_k^{[i]}$, $k \in \{1 \dots q\}$.
For $k = 1$ to q
    IF d_k is integer (d^k is the direction returned by $CSP_k^{[i]}$) THEN
        Set $\bar{x}^{[i]} = \bar{x}^{[i]} + d^k$.
    ELSE
        Set $P^{[i]} = P^{[i]} \cup \{j : d_j^k > 0\}$
    ENDIF
End For.
IF some d_k is fractional, construct RP according to $P^{[i]}$ and solve it by a MIP solver.
Send the resulting $\bar{x}^{[i]}$ and duals $\pi_k^{[i]}$, $k \in 1..q$ to the master agent.

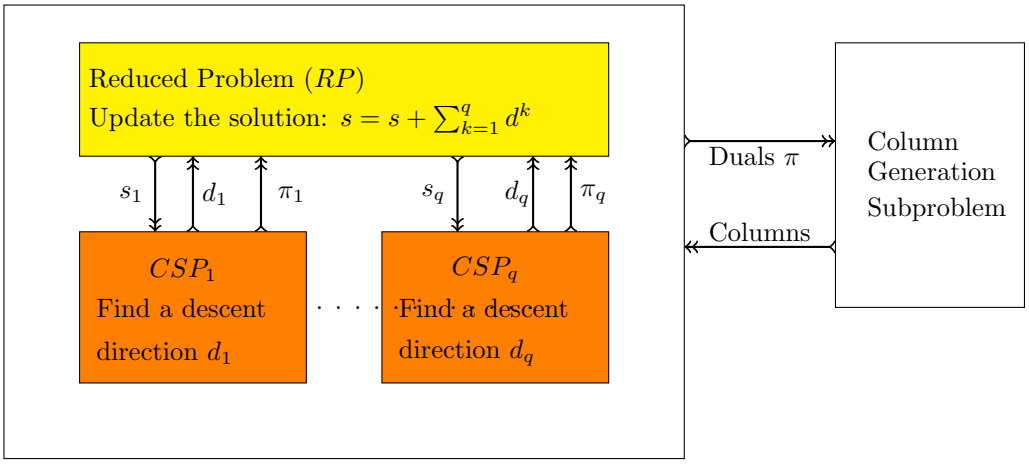

Figure 3: Column generation in DVD mode

3.1.3 IVD mode

The IVD can be seen as a neighborhood exploration policy. The idea is to partition columns of the constraint matrix A into subsets. In Foutlane et al. (2017, 2018), we used reduced cost as a pricing criterion. One may use other criteria such as the incompatibility degree. Thus, a worker agent starts by a neighborhood containing potential columns according to some chosen criteria and explores the subsets of columns incrementally. More precisely, the worker agent runs Zoom on SPP_k, a restriction of SPP to the chosen subset of columns, starting with the best solution that is returned by the master agent. Algorithm 3 provides the pseudocode of the IVD phase; q' is a parameter tuned by experimentation. An illustration is given in Figure 4.

Algorithm 3 IVD pseudocode for agent i

Price out the columns using a distance metric to create neighborhoods.
Sort the variables in an increasing order of the distance and reindex them.
For $k = 1$ to q'
    Build $\text{SPP}_k^{[i]}$ by considering the first $k\frac{|J|}{q'}$ variables.
    Solve $\text{SPP}_k^{[i]}$ with Zoom, set $\bar{x}^{[i]}$ to the obtained solution, and update $z_{ub}^{[i]}$.
    Send $\bar{x}^{[i]}$, calculate a dual vector $\pi^{[i]}$ and send it to the master agent.
End for.

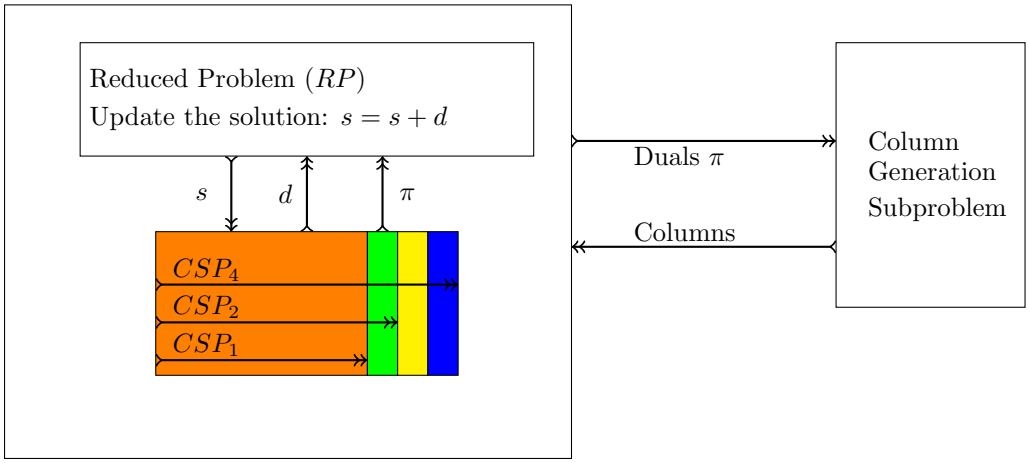

Figure 4: Column generation in IVD mode for q'=4

3.2 Master agent

The master agent controls the progress of DICG. It ends it when a termination criterion is satisfied. Similarly to the work of Foutlane et al. (2018), we develop and study two variants of DICG: the cooperative variant where worker agents cooperate and the competitive variant where worker agents work independently. Pseudocode of cooperative and competitive variants are provided in Sections 3.2.2 and 3.2.1 respectively. The IN-PARALLEL and END-PARALLEL terms are used to mention that the multiple statements in between are executed in parallel. For both variants, the master agent starts with an initial primal solution x_0 of value z_0 and a dual vector π_0, sets the upper bound $z_{ub} = z_0$, sends supp(x_0) and π_0 to all agents and waits for the pairs (solution, dual) obtained by the worker agents.

The master agent solves in parallel the linear relaxation of SPP by column generation. During a certain number of iterations (IterWarm), it sends the dual solution π_b of its restricted master poblem to the workers for a warm up. After the warm up phase, when the master receives a solution $\bar{x}^{[i]}$ from the i^{th} agent that improves the DICG upper bound z_{ub}, the master agent updates z_{ub} and x_b, the best solution encountered. During the DICG execution, the master agent reacts according to which variant, competitive or cooperative, is activated. Finally, the master agent initializes the counter $nbrItr^{[i]}$ for each agent and increments it after receiving a solution from the worker agent i. The master agent uses this counter to tell the worker which mode to use, GEN, DVD or IVD mode, depending on the predefined values IterDVDMax, IterWarm and ϵ_{dvd}. In addition, this counter is used among others to stop the agent i when it reaches a predefined value $IterMax$.

3.2.1 DICG competitive agents

After the warm-up phase, which aims to generate a sufficient number of columns, the worker agents begin to solve their RMPs using DISUD. They send their solutions to the master. The latter updates the i^{th} agent's upper bound $z_{ub}^{[i]}$. The sender agent makes another DVD iteration if its iteration number does not exceed a predefined value IterDVDMax and the current solution quality is less than a predefined value ϵ_{dvd}. Otherwise, the master agent sends to the i^{th} agent the message msgMODE-IVD in order to switch to IVD iteration. An agent becomes idle when its iteration number exceeds $IterMax$. Finally, DICG stops when all worker agents become idle or some stopping criteria are met. As it can be seen, each agent works independently and does not share any information with other agents. Algorithm 4 presents the master competitive procedure.

Algorithm 4 Competitive pseudocode

Set $z_{ub} = z_0$, $x_b = x_0$, $\pi_b = \pi_0$ and for each agent i, set $nbrItr^{[i]} =$ IterWarm, $mode^{[i]} = GEN$
IN-PARALLEL
  Calculate a lower bound z_{lb} for SPP and update π_b consequently. During the first IterWarm iterations, send msgGEN and π_b to all worker agents.
  Listen to worker agents:
    On the reception of a solution $\bar{x}^{[i]}$ from some agent i, DO
      Set $nbrItr^{[i]} = nbrItr^{[i]} + 1$, $\pi_b = \pi^{[i]}$
      IF $c \cdot \bar{x}^{[i]} < z_{ub}$ THEN
        Set $z_{ub} = c \cdot \bar{x}^{[i]}$ and $x_b = \bar{x}^{[i]}$
      END IF
      IF $nbrItr^{[i]} = IterMAX$ THEN
        Send msgMODE-IDLE to agent i, set $mode^{[i]} = IDLE$
      ELSE IF $nbrItr^{[i]} \leq IterDVDMax$ AND $\frac{z_{ub}-z_{lb}}{z_{lb}} > \epsilon_{dvd}$ THEN
        Send msgGEN and π_b to agent i
        Send msgMODE-DVD to agent i
      ELSE
        Send msgGEN and π_b to agent i
        Send msgMODE-IVD to agent i
      END IF.
      IF all worker agents are IDLE or some stopping criteria are met THEN
        Send msgSTOP to all worker agents and return x_b
      END IF.
    END DO
END IN-PARALLEL

3.2.2 DICG cooperative agents

In the cooperative variant, the master agent intervenes more and changes the worker agents settings during the progress of the process as it is shown in Algorithm 5. We discuss below the most important issues. During the process execution, the communication between the master and the worker agents is bilateral: the worker agent sends its newly found solution pair (primal, dual) to the master and waits for the primal solution x_b from which it starts, the OK to stay in DVD mode or any other decision from the master. Here again, when an agent iteration number exceeds $IterMax$, its mode changes to IDLE. If all agents are idle, the master stops the process. We limit the cooperation to the exchange of the best integer solution found at the end of a column generation iteration between the worker agent and the master agent. But, we keep in mind that further cooperation policies and strategies can be made in future work to study more thoroughly this subject.

Algorithm 5 Cooperative pseudocode

Set $z_{ub} = z_0$, $x_b = x_0$, $\pi_b = \pi_0$ and for each agent i, set $nbrItr^{[i]} =$ IterWarm, $mode^{[i]} = GEN$
IN-PARALLEL
  Calculate a lower bound z_{lb} for SPP and update π_b consequently. During the first IterWarm iterations, send msgGEN and π_b to all worker agents.
  Listen to worker agents:
    On the reception of a solution $\bar{x}^{[i]}$ from some agent i, DO
      Set $nbrItr^{[i]} = nbrItr^{[i]} + 1$, $\pi_b = \pi^{[i]}$
      IF $c \cdot \bar{x}^{[i]} < z_{ub}$ THEN
        Set $z_{ub} = c \cdot \bar{x}^{[i]}$ and $x_b = \bar{x}^{[i]}$
      ELSE
        SEND msgSOL and supp(x_b) to agent i, set $x^{[i]} = x_b$
      END IF
      IF $nbrItr^{[i]} = IterMAX$ THEN
        Send msgMODE-IDLE to agent i, set $mode^{[i]} = IDLE$
      ELSE IF $nbrItr^{[i]} \leq IterDVDMax$ AND $\frac{z_{ub}-z_{lb}}{z_{lb}} > \epsilon_{dvd}$ THEN
        Send msgGEN, π_b and msgMODE-DVD to agent i
      ELSE
        Send msgGEN, π_b and msgMODE-IVD to agent i
      END IF
      IF all worker agents are IDLE or some stopping criteria are met THEN
        Send msgSTOP to all worker agents and return x_b
      END IF.
    END DO
END IN-PARALLEL

4 Computational results

In this section, we present results of DICG and discuss its effectiveness. We tested DICG algorithm on crew pairing problem (CPP) and vehicle and crew scheduling problem (VCSP) instances. We compare DICG to DRMH, a distributed version of the well-known restricted master heuristic (RMH) (see Joncour et al. (2010)). DRMH consists in solving the linear relaxation of MP by column generation at the root node. Then, DCPLEX, the distributed version of CPLEX, solves the last RMP after adding integer constraints on the variables.

At the beginning, we go through characteristics of the instances composing our test benchmark. Then, we discuss the influence of the two parameters IterWarm, which controls the duration of the warming phase, and q, the number of complementary problems, on the performance of DICG. After this, we compare DICG$_{comp}$ and DICG$_{coop}$ performances when using the best values of these parameters. Finally, we compare the best variant of DICG to DRMH.

We implemented the two DICG variants (competitive (DICG$_{comp}$) and cooperative (DICG$_{coop}$)) using C^{++} and the MPI (Message Passing Interface) library. This latter ensures communication between our agents. The master runs on a Linux PC with Quad-Core processor of 3.30 GHz and each worker agent runs on a Linux PC with 8 processors of 3.4 GHz each. Finally, we note that all induced optimization problems (CSP, RP, RMP ...) are solved using the commercial CPLEX solver version 12.6.1 while the SPPRC is solved using the Boost library version 1.55.

4.1 Instances characteristics

The set of tests consists of CPP and VCSP instances, described respectively in Sections 4.1.1 and 4.1.2. Each subset contains medium and large instances. Furthermore, considering that DICG needs a pair (primal, dual) of solutions to start from, we construct an artificial initial primal solution where each task is covered by a single-task column with a large, big-M, cost and an initial dual solution where each dual value is set to this large cost value.

4.1.1 CPP instances

In aircrew scheduling, a pairing is a sequence of flights that starts and ends at the same airport. CPP consists of finding a set of pairings that covers all the scheduled flights at minimum cost over the planning horizon. Moreover, each flight has to be covered by a single pairing and therefore, CPP is modeled as a SPP. In practice, the CPP is solved by branch & price method where the pairings are generated by solving subproblems modeled as SPPRCs (see Saddoune et al. (2013)). For our tests, we use five instances derived from a real-life CPP of a major north American airline. The original datasets can be found in Kasirzadeh et al. (2017) (aircraft fleets concerned are D94, D95, 757, 319, and 320). We add the CPP prefix to the instance name to indicate that it belongs to the CPP subset. Table 1 presents the CPP instances characteristics. This table shows (from left to right) the name of the instance, the number of tasks (flights), the "density" (the rounded average number of flights per pairing (i.e., the number of nonzeros per column)), the (approximate) percentage of degeneracy in an optimal basis.

Table 1: Characteristics of the CPP instances

instance	nbrTasks	density	degeneracy
CPP_D94	424	8	87
CPP_D95	1255	9	88
CPP_757	1290	6	83
CPP_319	1293	7	85
CPP_320	1740	7	85

4.1.2 VCSP instances

VCSP consists of assigning buses to bus trips and drivers to tasks which are defined by dividing each bus trip into segments. These latter link consecutive relief points where drivers exchange could occur. There is one

task for each (bus trip) segment. We consider the single-depot homogenous-fleet variant addressed by Haase et al. (2001) and consider only set partitioning constraints. We use the random instance generator of Haase et al. (2001) to generate three test instances for the same pair (R, B) where R is the number of relief points on each bus trip and B is the number of bus trips. In fact, the size of an instance (number of tasks) is defined as the product $B \times (R+1)$. Consequently, to name a VCSP instance, we use the acronym vcs_s_R_B. The prefix *vcs* indicates that it belongs to the VCSP subset and the incremental value s indicates the seed number of the instance. So, as an example, vcs_1_5_160 indicates that it is the second VCSP instance generated with the values s = 1, R = 5, and B =160. Table 2 presents the VCSP instances characteristics. The first column shows the name of the instance. Then, it reports the number of tasks (nbrTasks), the number of relief points (R), the number of bus trips (B), the density (density), and the percentage of degeneracy (degeneracy).

Table 2: Characteristics of the VCSP instances

instance	nbrTasks	R	B	density	degeneracy
vcs_s_9_80	800	10	80	12	91
vcs_s_5_160	960	6	160	9	89
vcs_s_9_160	1600	10	160	15	93
vcs_s_9_200	2000	10	200	14	93

4.2 Influence of the parameters

In this section, we study the influence of the principal parameters, i.e., the number q of CSPs we solve in parallel and the number of column generation iterations (IterWarm) in the warm up phase. We present results of the influence of these parameters on the CPP_319 instance. The idea is to tune these parameters on a medium instance (leaning towards large ones) and use the tuned values for all the other instances.

First, as mentioned in Section 3.1.2, the behavior of an agent is defined by the scoring function we and the normalization weight vector w. The workers that we used during our tests are the following three agents:

- agent 1 defined by the pair (w_1, we_1),
- agent 2 defined by the pair (w_2, we_1), and
- agent 3 defined by the pair (w_3, we_1).

These agents use the same scoring function we_1 - as it seems to be the best for our tests - and different normalization weight vectors. They all look for descent directions in a region of the graph G where there are more negative reduced cost columns, but differ only on how the direction is composed. Indeed, the first agent favors directions having the minimum average reduced cost per entering column, the second favors entering columns with small incompatibility degree, while the third leans towards entering columns covering few tasks.

Figure 5 shows the influence of q on the evolution of the objective value that DICG finds over time during DVD phase. We choose the values of q so that they are power of 2 (q = 2, 4, 8). We deduce that we get good performance for both $q = 2$ and $q = 4$. This can be explained by the fact that for high values ($q = 8$), we get poor performance because it becomes difficult to find descent direction as more variables are ousted by the DVD decomposition process. In addition, when there are more processes, we face the overload phenomenon as we use computers with 8 processors.

Similarly, Table 3 gives results for the influence of IterWarm. The first column shows the different tested values of IterWarm parameter. Then, for each DICG variant, it reports the total computational time in seconds (time), the optimality gap (in percentage) between the cost of the best solution found and the linear relaxation optimal value (gap), the number of column generation iterations (nbIter) and the agent that gets the best solution (Ag_b). The IterWarm parameter controls the duration of the warming phase. Indeed, higher is the IterWarm value, higher is the warming up time. From the results, we deduce that the DICG performance increases with IterWarm value. In general, DICG variants perform well. This is explained by the fact that ISUD and its variants perform better when they generate columns close to LP optimality and the integrality gap is small.

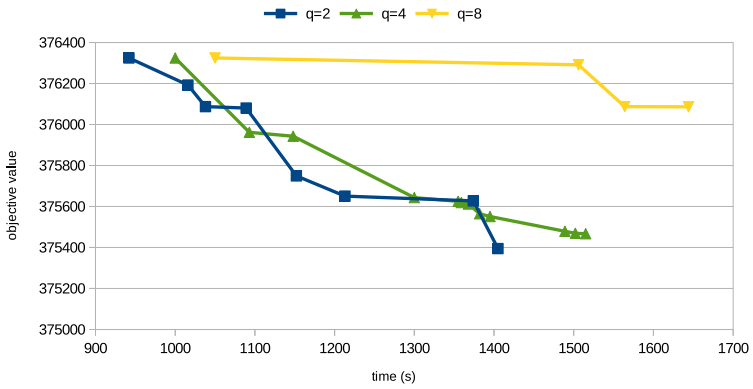

Figure 5: Influence of q on CPP_319 during the DVD phase

Table 3: Influence of IterWarm on CPP_319

	DICGcomp				DICGcoop			
IterWarm	time	gap (%)	nbIter	Ag_b	time	gap (%)	nbIter	Ag_b
5	5604	0.95	35	2	6786	0.62	44	3
10	3547	0.37	29	1	5399	0.29	16	2
15	2579	0.19	26	2	2879	0.23	25	1
20	5896	0.17	26	2	4148	0.20	41	3

Based on these results, we can see that we get good quality solutions in shorter computing times for IterWarm = 15. Therefore, we use the values $q = 4$ and IterWarm = 15 for the results given in sections 4.3 and 4.4. Moreover, we set $\epsilon_{dvd} = 2\%$; the threshold of 2% is inspired by an industrial observation claiming that solutions within 2% gap are acceptable in practice (see Rosat et al. 2017b). Also, we set the other DICG parameters as follows: IterDVDMax = 20, IterMAX = 50 and the execution time limit to two hours for CPP instances and to a one hour for VCSP instances.

4.3 Cooperative vs competitive results

In this section, we show DICG_{coop} and DICG_{comp} results and discuss their performances on our set of tests. Figure 6 shows the evolution of the objective value over time for DICG_{comp} on the instance CPP_320, as it is the largest instance. It depicts the solutions found by each agent during the solution process.

We can observe a rapid objective value decrease at the beginning of the solution process compared to the objective value decrease at the end. This is explained by the fact that like traditional column generation methods, it becomes difficult for DICG to generate improving directions when the solution process approaches the optimality. This behavior is typical and representative of the other instances.

Figure 7 shows the evolution of DICG_{coop} and DICG_{comp} on CPP_320. We connect the points to improve its readability. We note that the two curves present similar shape. DICG_{coop} is better in the middle of the solution process. This is due to the fact that DICG_{coop} embeds the spirit of the depth first search strategy. DICG_{coop} uses all its agents to explore its best solution x_b (solution with the lowest cost) neighborhood.

Table 5 shows results for each variant of DICG. It reports the same information as in Table 3 for the columns having the same name. In addition, it presents the objective value (obj) and the number of improving integer solutions found during the solution process (nbSols).

We observe that both DICG variants solve all instances to near optimality within almost an hour. The differences between objective values of the two variants are less than 0.1% in all cases. Thus, the results show that both variants are excellent from industrial point of view. However, DICG_coop remains significantly better in terms of the number of integer solutions found and the overall objective value thanks to cooperation. On the other hand, concerning the computing time, DICG_{comp} is faster than DICG_{coop} because the marginal

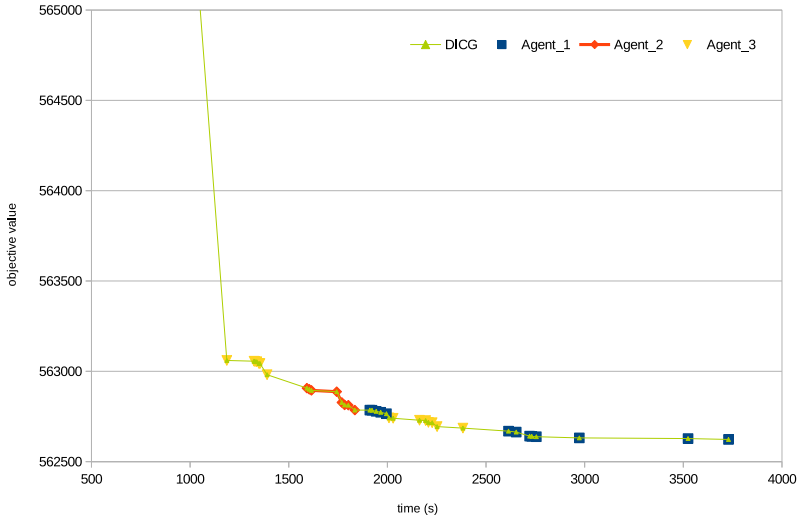

Figure 6: DICG$_{comp}$ evolution over time on CPP_320

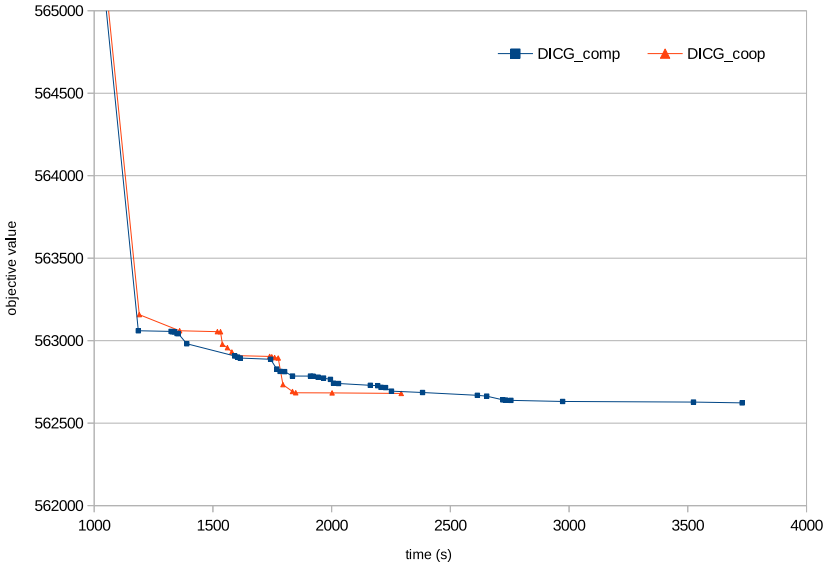

Figure 7: DICG variants evolution over time of the CPP_320 instance

Table 4: Results of DICG variants for the CPP instances

instance	DICG$_{comp}$					DICG$_{coop}$				
name	time	obj	nbIter	nbSols	Ag$_b$	time	obj	nbIter	nbSols	Ag$_b$
CPP_D94	59	109178	17	2	2	59	109178	17	2	2
CPP_D95	3296	268975	19	57	3	4580	268431	25	95	1
CPP_757	708	430807	16	5	3	1457	430671	14	19	1
CPP_319	2579	374859	26	35	3	2879	375015	25	46	1
CPP_320	3729	562623	25	36	1	2293	562680	18	17	1

gain in the last iterations is costly. In major complex applications, it worths it, especially when we have enough time for planning. Finally, it is obvious that all agents contribute to the DICG solution process as it is shown by the Ag_b column. Based on this, we deduce that considering many agents simultaneously is a better approach.

From the aforementioned results, we conclude that DICG variants yield excellent results. DICG$_{coop}$ constitutes a good variant of DICG that shows a good potential since it allows to manage multiple agents simultaneously in order to take advantage of their cooperation and paves the way for implementing more sophisticated cooperation strategies.

4.4 DICG vs DRMH

The goal in this section is to compare the performance of the best variant of DICG against DRMH on CPP and VCSP instances. Figure 8 shows the gap value evolution over time for DICG variants and DRMH on CPP_320. The figure clearly shows that the DICG variants outperform DRMH in terms of computing time and solution quality.

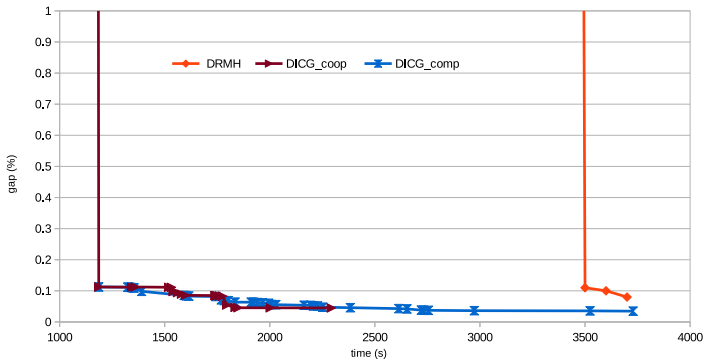

Figure 8: Gap evolution for DICG and DRMH for CPP_320 test

The CPP computational results are presented in Table 5. We report (from the left to the right) the computational time, the solution gap, the total number of improving integer solutions and the best DICG variant. We observe that for the cited statistics, DICG is generally better than DRMH, especially for the number of improving integer solutions that is very desirable in practice. We would like to mention here that DRMH is good because the density is low which is an important fact.

Table 5: Results of DICG and DRMH for the CPP instances

instance	DRMH				DIGC				
name	time	gap (%)	nbrSol	nbIter	time	gap(%)	nbrSol	nbIter	bestVar
D94	81	0.07	9	22	59	0.35	2	17	coop
D95	4110	0.71	9	36	3296	0.48	95	25	coop
757	1392	0.05	4	30	1457	0.04	18	19	coop
319	6649	0.28	12	40	2879	0.23	46	25	coop
320	2946	0.03	6	42	3729	0.03	36	25	comp

The VCSP computational results are presented in Table 6. We report the computational time, the solution gap and the number of column generation iterations for both DRMH and DICG. We restrict the comparison of VCSP instances to the DICG_comp variant. This is dictated by the fact that the convergence of DICG is too fast that there is no need to any cooperation strategy between worker agents. For this reason again, we do not report results on VCSP instances in the previous section. For instances with medium difficulty, i.e., those with tasks' number less than 1000 and low density, DICG is two to five times faster than DRMH. While for instances with larger number of tasks' number (more than 1000) and higher density (greater than 13), DRMH is unable to find a good enough (less than 10% gap) feasible integer solution within one hour time limit.

Remark 1 *This can be explained by the fact that in DRMH, columns that are good for the linear relaxation are not necessarily good for getting an optimal integer solution. In the opposite, we can show that at each CG iteration, DICG should succeed in generating one or more optimal columns that are missing in the current*

integer solution. This can also explain the fact that the DICG number of iterations is smaller than the DRMH number of iterations number.

Table 6: Results of DICG and DRMH on the VCSP instances

instance	DRMH			DICG			
	time	gap (%)	nbIter	time	gap (%)	nbIter	Ag_b
vcs_0_9_80	73	0	37	44	0	3	3
vcs_1_9_80	105	0	10	38	0	3	2
vcs_2_9_80	74	0	38	39	0	2	2
vcs_0_5_160	123	0	37	25	0	3	2
vcs_1_5_160	119	0	40	26	0	4	3
vcs_2_5_160	112	0	29	24	0	4	3
vcs_0_9_160	3600	-	-	163	0	4	3
vcs_1_9_160	3600	-	-	154	0	4	3
vcs_2_9_160	3600	-	-	151	0	4	1
vcs_0_9_200	3600	-	-	220	0	4	1
vcs_1_9_200	3600	-	-	220	0	5	1
vcs_2_9_200	3600	-	-	212	0	6	3

5 Conclusion

We proposed in this paper a new algorithm DICG, which is a distributed integral column generation algorithm. It is a multi-agent based algorithm dedicated to generate in parallel descent directions leading to an improved integer solution at each column generation iteration. We presented and implemented two DICG variants and discussed their performances. They differ in the strategy used to manage the worker agents. We showed that our algorithm yields good quality solutions (less than 1%), largely better than the distributed version of RMH on a set of vehicle and crew scheduling instances. Our tests set contains large-scale instances with up to almost 2000 tasks. DICG was able to find optimal or near optimal solutions for all instances in less time than DRMH, especially on hard VCSP instances.

Future research should be done to further improve the DICG performance. We believe that combining this primal algorithm DICG with meta/math/heuristics should produce better solutions in a drastically reduced time.

References

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

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

G. Dantzig and P. Wolfe. Decomposition principle for linear programs. Operations Research, 8:101–111, 1960.

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.

G. Desaulniers, J. Desrosiers, and M. Solomon. Column generation. Springer, New York,, 2005.

M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. Transport Science, 23:1–13, 1989.

I. El Hallaoui, A. Metrane, F. Soumis, and G. Desaulniers. An improved primal simplex algorithm for degenerate linear programs. 2011.

O. Foutlane, I. El Hallaoui, and P. Hensen. Integral simplex using double decomposition. Cahiers du Gerad, G–2017–73, HEC Montreal, 2017.

O. Foutlane, I. El Hallaoui, and P. Hensen. Distributed integral simplex for clustering. Cahiers du Gerad, G–2018–31, HEC Montreal, 2018.

M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A column generation approach for large-scale aircrew rostering problems. Operations Research, 47(2):247–263, 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.

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

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

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

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

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

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

S. Rosat, I. Elhallaoui, F. Soumis, and A. Lodi. Integral simplex using decomposition with primal cutting planes. Mathematical Programming, 2017a.

S. Rosat, F. Quesnel, I. Elhallaoui, and F. Soumis. Dynamic penalization of fractional directions in the integral simplex using decomposition: Application to aircrew scheduling. European Journal of Operational Research, 263: 1007–1018, 2017b.

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

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

A. Zaghrouti, I. El Hallaoui, and F. Soumis. Improving set partitioning problem solutions by zooming around an improving direction. Annals of Operations Research, Apr. 2018.