Les Cahiers du GERAD ISSN: 0711–2440

Distributed integral simplex for clustering

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

G–2018–31

April 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

Distributed integral simplex for clustering

Omar Foutlane [a, b]

Issmail El Hallaoui [a, b]

Pierre Hansen [a, b]

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

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

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

April 2018
Les Cahiers du GERAD
G–2018–31

Abstract: Clustering is the subject of active research in several fields such as operations research, statistics, pattern recognition, and machine learning. The range of applications is very wide: scheduling, vehicle routing, pattern recognition, etc. Depending on the specific needs of the community, the methods used to solve these problems vary from heuristics of rather primal nature (improving of clustering iteratively by relocation moves for instance) to exact methods of a rather dual nature where we generally solve the continuous relaxation by releasing the integrality constraints and restoring it by implicit enumeration (branch and cut or branch and cut and price). In this paper, we propose the integral simplex, an exact primal method that could be suitable for both major classes. More interestingly, it could be distributedly solved better than the dual approach. Consequently, this work aims to propose a distributed version of the integral simplex, called DISUD, using some decompositions and multiple agents. Each agent dynamically splits the overall set partitioning (clustering) problem into sub-problems and solve them in parallel on a single machine. The new algorithm DISUD improves at each iteration the current clustering until (near) optimality is reached. It works much better on real set partitioning instances from the airline industry than DCPLEX, the distributed version of the state of the art commercial solver CPLEX.

Keywords: Set partitioning problems, integral simplex using decomposition, multi-agents systems, distributed processing techniques

1 Introduction

Clustering is the subject of active research in several fields such as operations research, statistics, pattern recognition, and machine learning. Set partitioning problem (SPP) is a combinatorial optimization problem that models well many interesting real-life clustering problems. The range of applications is very wide: scheduling, vehicle routing, pattern recognition, etc.

Depending on the specific needs of the community, the methods used to solve these problems vary from heuristics of rather primal nature (improving of clustering iteratively, by relocation moves for example) to exact methods of a rather dual nature as classified by Letchford and Lodi 2002 (solving the continuous relaxation by releasing the integrality constraints and restoring it by implicit enumeration (branch and cut or branch and cut and price) when the clustering is too much constrained, like in aircrew scheduling. For instance, in the latter there are too many safety and collective agreement rules we have to respect to cluster flights (objects), i.e., flights scheduled for a pilot. In this paper, we propose the integral simplex, an exact primal method that could be suitable for both major classes. More interestingly, it could be distributedly solved better than the dual approach.

Despite our focus on the vehicle and crew scheduling applications (including but not limited to truck deliveries (see Balinski and Quandt 1964), vehicle scheduling (see Ribeiro and Soumis 1991), aircrew and bus driver scheduling (see Desaulniers et al. 1994, Chu et al. 1997, Hoffman and Padberg 1993)), the concepts and methods outlined in this paper are theoretically usable for the other application contexts.

SPP can be defined using the following crew scheduling applications 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:

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

$$(SPP) \quad subject\ to$$

$$\sum_{j \in J} a_{tj} x_j = 1, \forall t \in T \tag{2}$$

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

The objective function (1) seeks to minimize the total cost. The set partitioning constraints (2) ensure that each task is covered exactly once. Constraints set (3) imposes integrality on the x_j variables. The linear relaxation (LP) is obtained by replacing (3) by $x_j \geq 0, \forall j \in J$. An optimal solution of SPP consists of selecting a subset of schedules such as each task is done by one and only one schedule and the sum of the costs of the subset schedules is minimized.

The remainder of this paper is organized as follows. The literature review is presented in Section 2 and an overview of the contributions in Section 3. Section 4 presents briefly some useful preliminaries on the decomposition basics and the main parts of Zoom. Section 5 describes the new algorithm DISUD and provides a detailed algorithmic and theoretical analysis of its components. In Section 6, 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 7.

2 Literature review

SPP is NP-hard (see Garey and Johnson 1979). Given the wide use of SPP, many heuristic and exact algorithms have been developed to solve it. We focus on exact algorithms, using possibly heuristic stopping criterion that guarantees in practice a certain quality of the solution (optimality or near optimality). The most known method is the famous branch and cut (Hoffman and Padberg 1993, Desaulniers et al. 1997). However, this method becomes inefficient and takes huge time to reach an optimal solution for large instances due to degeneracy and the "explosion" of the branching tree.

SPP degeneracy complicates too much the solution of large $SPPs$. To deal with degeneracy in LP, El Hallaoui et al. 2011 presented the improved primal simplex (IPS) based on a decomposition processus. They decompose the problem into a non-degenerate easy to solve reduced problem and a complementary problem that finds descent directions to escape local optima of the reduced problem. Recently, Zaghrouti et al. 2014 developed the integral simplex using decomposition (ISUD), which is based on IPS, to solve SPP (with integrality constraints). 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, other research works have been done by Rosat et al. 2016, 2017a and Zaghrouti et al. 2013 in order to improve ISUD performance. Rosat et al. 2017a studied the impact of adding cuts to ISUD and finds that this technique is costly in time computing for large instances. In addition, Rosat et al. 2016 compared different normalisation constraints of the cone of directions used by ISUD. As for Zaghrouti et al. 2013, they developed the Zoom algorithm based on ISUD. It explores a neighborhood of the current integer solution when it is not possible to find an improving "integer" direction, i.e., leading to an improving integer solution, using ISUD, see Section 4.2 for more details. This neighborhood is constructed using the "fractional" direction returned by the complementary problem.

Nowadays, there is an increased interest to use parallel and a fortiori distributed algorithms especially with the advent of parallel computers in the world of scientific computing. The aim is to improve the solution time and to increase the size of the treated problems. Bürger et al. 2012 introduced an interesting distributed algorithm to solve degenerate linear problems. Their idea is to split the columns over a certain number of machines (agents). Each one solves the resulting reduced problem. The agents exchange their optimal bases and continue solving until the bases are all the same.

Except distributed Branch & Bound and Branch & Cut, distributed algorithms dedicated to primal integer programing and especially to SPP are to the best of our knowledge inexistant. Indeed, many authors have worked to develop distributed versions of Branch & Bound and Branch & Cut (see Eckstein 1993, Laursen 1993, Quinn 1990, Fischetti et al. 2018). They discuss issues such as architecture and communication. They apply distributed computing techniques to Branch & Bound and Branch & Cut mainly by distributing the computation of the branching tree of subproblems over multiple nodes (machines). Those research works have led to many applications. We refer the reader to an early survey by Gendron and Crainic 1994 and to the more recent one by Ralphs et al. 2017. In the latter, we report the main and classical issues that still persist: disproportionate amount of time spent in the shallowed nodes (root node particularly), unbalanced search tree, dynamic construction of the tree, dynamic generation of useful information (cuts, bounds), and consequently the need to some synchronization to avoid redundant work, which leads generally to a worse performance and scalability. CPLEX implements such a distributed mechanism using the Supervisor-Worker scheme, a kind of Master-Worker scheme where the master stores no data concerning the search tree. Its role only is to coordinate the load balancing.

The trend to develop distributed algorithms and the interest aroused by ISUD motivates us to look for a distributed version of it. The idea is to invest parallelizing a primal approach instead of a dual approach (the branch and bound for instance). This resolves the classical issues raised above. Foutlane et al. 2017 developed the integral simplex using double decomposition algorithm (ISU2D) which we generalize and improve ISU2D in this paper. ISU2D is a parallel variant of the ISUD. ISU2D splits the original problem into small subproblems and solves them to get an improved solution at each iteration. The authors showed the existence of an optimal decomposition which leads to an optimal solution. The authors proposed an iterative procedure for finding such decomposition and showed the potential of such parallel methods.

3 Contributions overview

In this paper, we use multi-agent system approach (MAS) to introduce a general framework for a distributed version of ISU2D called DISUD. We consider a network of worker agents, rather than a single agent such as in ISU2D (see Foutlane et al. 2017), that split SPP into a set of small subproblems solved in parallel, each of which containing a niche of potential improving columns. We summarize below the most important contributions of the paper:

- DISUD develops a procedure that can be seen as a parallel adaptation of *Zoom* to quickly improve the worker solution. Instead of zooming around one direction like in Zaghrouti et al. 2013, the procedure does a *multizoom* by zooming around a multitude of orthogonal directions.
- We present new theoretical results. We show that the proposed decomposition is, contrary to ISU2D, less sensitive to dual values; it depends only on the costs of the current solution to improve. We also show that increasing the number of processors beyond a certain limit (the diameter of the polytope) is useless.
- We compare two versions: one competitive (a kind of racing implementation) and another cooperative that exploits the information gathered during the solution process. At each iteration, the two versions use dynamic decompositions simultaneously (in parallel) to guide the search to improved integer solutions.
- Tests on real crew pairing instances from the airline industry, with up to 1740 flights (per week) and more than one million of variables, show the effectiveness of DISUD compared to DCPLEX, the distributed version of CPLEX, which is the state of the art commercial solver. We succeed to compute better quality solutions (often optimal or near-optimal) in much less computational time for most instances.

4 Preliminaries

In this section, we provide the basic notions necessary to understand DISUD. We present the decomposition principles and the main parts of the Zoom algorithm. A parallel version of the latter is used by the agents of DISUD to solve their image of SPP. Zoom proved to be efficient in practice.

4.1 Decomposition basics

Given an integer solution $\bar{x}$ to SPP, let P_{int} be the index set of its positive components, i.e., $P_{int} = supp(\bar{x}) = \{j \in J : \bar{x}_j = 1\}$, P an index set of some linearly independent columns containing at least P_{int}, and $p = card(P)$. 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 *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.*

Let C and I be the index sets of the compatible and incompatible columns respectively. Thus, J is partitioned into C and I, i.e., $J = C \cup I, C \cap I = \emptyset$. RP is defined as a restriction of SPP to compatible columns only:

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

As $p \leq m$, there could be some redundant constraints that we should remove from RP. When $P = P_{int}$, it is interesting to see 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. Observe that $\bar{x} = (x_C^*, 0)$ is a solution to SPP.

To improve $\bar{x}$, we use a complementary problem CP to find a set of incompatible columns to replace a subset of the current solution columns. More precisely, we look for a (convex) combination of incompatible columns that is compatible and has a negative reduced cost. CP can be formulated as:

$$\begin{aligned} & \textit{Minimize} && \sum_{j\in I} c_j v_j - \sum_{l\in P} c_l \lambda_l && (7)\\ (CP) \quad & s.t. && \sum_{j\in I} A_j v_j - \sum_{l\in P} A_l \lambda_l = 0 && (8)\\ &&& e \cdot v_I = 1 && (9)\\ &&& v_j \geq 0 \,, j \in I && (10) \end{aligned}$$

We can easily show that $d = (v, -\lambda, 0)$ defines a descent direction. Zaghrouti et al. (2013) 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 SPP. Otherwise, CP guarantees to find a descent direction leading to an improved integer solution. 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. If 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 obtain a descent direction leading to an improved integer solution. In this case, S^+ is shown to be minimal by El Hallaoui et al. (2011), i.e., nondecomposable using the terminology of Balas and Padberg (1975), meaning that pivoting on variables indexed by S^+ leads to an adjacent extreme point with improved cost.

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. A_P^2 is of course composed of dependent rows. Similarly, let $A_I = \begin{pmatrix} A_I^1 \\ A_I^2 \end{pmatrix} = (a_{ij})_{\substack{1\leq i\leq m \\ j\in I}}$ be a submatrix of A composed of incompatible columns indexed by I with A_I^1 a $|P| \times |I|$ matrix. The variables λ can be eliminated by using the fact that the columns of A_P are linearly independent. 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 could be replaced. This results in the Equation (4.1).

$$\begin{aligned} (CP) \qquad & z_P^{CP} = \min_v && \left(c_I^\top - c_P^\top \left(A_P^1\right)^{-1} A_I^1\right) v && (11)\\ & \text{s.t.} && \left(A_P^2 \left(A_P^1\right)^{-1} A_I^1 - A_I^2\right) v = \mathbf{0} && (12)\\ &&& e \cdot v = 1 && (13)\\ &&& v \geq \mathbf{0}. && (14) \end{aligned}$$

The incompatibility degree of a column A_j towards a given integer solution is a measure that represents a distance of the column from the solution. An example of this measure could be given by $\|MA_j\|_1$ where $M = (A_P^2(A_P^1)^{-1}, -I_{|P|})$. $I_{|P|}$ is the $|P| \times |P|$ identity matrix. This measure is equal to 0 for compatible columns and positive for incompatible ones. The constraint can be rewritten simply as: $MA_I v = 0$; A_I is the submatrix of A containing only columns indexed by I .

When columns A_j, $j \in S^+$ are not pairwise row-disjoint, the direction d is said to be fractional. Instead of branching in CP that is a little bit complicate due to the structure of CP, Zaghrouti et al. (2013) proposed to zoom around this "fractional" direction. We discuss this in the next subsection.

4.2 Zoom description

Zoom iterates between RP (compatible columns) and CP (incompatible columns) until it reaches an optimal solution. The main steps of Zoom 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 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.*

We mention here that no branching is done in CP. Actually, if the direction is fractional we construct RP around this direction as explained above and solve it by a MIP solver. Zaghrouti et al. (2013) reported that this RP has good properties: small gap and density, good initial solution (that is the current integer solution) to start from, easy to solve, big chances to get an improved integer solution. We report in Zaghrouti et al. (2013) that in more than 80% of the cases, the directions found were integer. That means that no MIP was solved in Step 2 in these cases. We simply set $\bar{x} = \bar{x} + |S^+|d$ because when the direction is integer we can show that positive entries of d are all equal: $v_j = \lambda_l = \frac{1}{|S^+|}$. The CP favors integrality by its nature. We refer the reader to Zaghrouti et al. (2014, 2013) for more details.

5 DISUD algorithm

As mentioned earlier, DISUD is a multi-agent algorithm where the master agent ensures, among others, the communication between other network agents called worker agents and some control stuff. Worker agents realize multiple decompositions of the problem and solve them in parallel. We implemented DISUD in a such a way that each worker does not have to wait for other agents to end their iteration to start a new iteration. Consequently, DISUD reduces overhead due to communication synchronization. Rather, it exploits the available time to improve the current solution. DISUD stops when the master agent receives a satisfactory solution.

In this section, we start by presenting the worker and master agents in detail in Sections 5.1–5.2 respectively. Finally, we give a theoretical analysis of DISUD in Section 5.3. We note that throughout this paper we use superscript [i] to denote quantities belonging to the i^{th} agent.

5.1 Worker agents

Worker agents realize multiple decompositions to increase the chance to get an improved solution. Each worker agent solves SPP using a specific decomposition either with DVD or IVD mechanisms as described briefly in Sections 5.1.1 and 5.1.2 respectively. More details are in Foutlane et al. 2017. A worker agent reacts to the messages received from the master as indicated in Algorithm 1 below. We then give an illustration using the MAS paradigm in Section 5.1.3.

Algorithm 1 Worker agent algorithm

```
Do
    Wait for a message from the master. In case of:
        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)
```

5.1.1 DVD mode

The idea of DVD is to split the original complementary problem into small "orthogonal" complementary subproblems (CSPs) that can be efficiently solved in parallel. In such subproblems, we look for replacing some of the variables from $supp(\bar{x}^{[i]})$ by some more interesting ones. So, during the DVD mode, the worker agent partitions $P_{int}^{[i]} = supp(\bar{x}^{[i]})$ into q clusters where the columns indexed by cluster k cover a set of tasks T_k, i.e., $T = \cup_{1\le k\le q} T_k$. Let $I_k \subset I$ be the subset of incompatible columns that cover only tasks in T_k. We have $I^{[i]} = \cup_{1\le k\le q} I_k$. The agent i constructs hence an RP and q complementary subproblems $(CSP_k^{[i]})_{1\le k\le q}$ formulated as follows:

$$CSP_k^{[i]} \qquad min \sum_{j\in J_k} \bar{c}_j v_j$$

$$MA_{I_k} v_k = 0 \tag{15}$$

$$\sum_{j\in I_k} v_j = 1 \tag{16}$$

$$v_j \in \{0,1\} \qquad \forall j \in I_k \tag{17}$$

To do so, each agent defines 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 $A_{v'}$. We define E as the set $\{(v,v') : I_{vv'} \neq \emptyset\}$. It is obvious that if $E = \emptyset$ then the constraint matrix A is a bloc diagonal matrix and SPP is composed of independent set partitioning problems that can be solved in parallel. This is generally not the case in practice.

Based on this, the weight of the edge $(v,v') \in E$ measures the "likelihood" that some of the variables indexed by $I_{vv'}$ could improve the objective value if entered into the basis. We partition the graph into q disjoint subgraphs using a min-cut algorithm (see Kernighan and Lin 1972). When the edge $(v,\ v')$ is not cut, A_v and $A_{v'}$ are grouped into a cluster and $\exists k$; $1 \le k \le q$ where $T_{vv'} \subseteq T_k$. Thus, the variables indexed by v and v' could be part of a descent direction (as leaving variables). Obviously, when the edge (v,v') is cut, it is not possible in the current iteration of DVD to improve the objective value with the variables indexed by $I_{vv'}$.

We proved in Foutlane et al. 2017 the existence of an optimal decomposition. But, as this latter is not known a priori, the worker agents implement different weighting methods simultaneously and consequently manage different weighted graphs $G(V,E)$ to increase the chance to find one rapidly. The efficiency of this depends heavily on the edge weights. Consequently, each agent of the network implements a different weighting method $we^{[i]}$ to calculate edge weights. We suggest computing $we^{[i]}$ as a function of the reduced cost, the incompatibility, the number of covered tasks (non zeros elements), and other relevant attributes of columns $A_j \in I_{vv'}$. In the following, we list four agents we used in this proof of concept. Two of them implement the two weighting methods that we studied in ISU2D. Of course, the list of agents is not exhaustive and other agents could be added easily using this framework.

Agent 1 . The first weighting method $we^{[1]}$ stipulates that a descent direction is more likely to exist in regions where there are more columns with negative reduced costs columns. We therefore cut the edges with the smallest number of negative reduced costs. Based on this, $we^{[1]}$ associates with each edge (v, v') the number of negative reduced cost columns that $I_{vv'}$ contains.

$$we^{[1]} : (v, v') \mapsto w_{vv'} = |\{j \in I_{vv'} \quad : \quad \bar{c}_j \leq 0\}|$$

Agent 2 . The second weighting method is inspired by the simplex algorithm. It 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. Therefore, $we^{[2]}$ associates with the edge (v, v') the absolute value of the smallest negative reduced cost column indexed by $I_{vv'}$.

$$we^{[2]} : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\bar{c}_j : \ j \in I_{vv'}\}).$$

Agent 3. The third weighting method $we^{[3]}$ derives from $we^{[2]}$. We compute $we^{[3]}$ using the reduced cost and the the number n_j of tasks covered by A_j. We stipulate that a good entering variable should have the smallest average negative reduced cost. Thus, when two columns have the same reduced cost $\bar{c}_j$, we favor the one that covers fewer tasks.

$$we^{[3]} : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\frac{\bar{c}_j}{n_j} : \ j \in I_{vv'}\}).$$

Agent 4. Given the importance of the degree of incompatibility in ISUD, we suggest to compute the fourth weighting method $we^{[4]}$ using the reduced cost and the incompatibility degree $k_j = \|MA_j\|_1$ of a column A_j. Indeed, the incompatibility degree can be seen as the distance from column A_j to the vector subspace generated by the columns of the current integer solution. It can be interpreted as the number of changes to the current solution. We compute $we^{[4]}$ then as:

$$we^{[4]} : (v, v') \mapsto w_{vv'} = -\min(0, \min\{\frac{\bar{c}_j}{k_j} : \ j \in I_{vv'}\}).$$

Observe that we thus favor solutions that are primally not too far from the current one because we suppose that the current one is good.

Hence, building the subproblems reduces to defining the partition $\tau = (T_k)_{1 \leq k \leq q}$. DVD procedure can be interpreted as a parallel adaptation of *Zoom* to quickly improve the worker agent solution. Instead of zooming around one direction, DVD does a *multizoom* by zooming around a multitude of orthogonal directions. Algorithm 2 outlines the DVD procedure.

Algorithm 2 DVD pseudocode for agent i

```
Build $\tau^{[i]}$ and consequently $CSP_k^{[i]}$, $k \in \{1 \dots q\}$ using $we^{[i]}$.
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 $\bar{x}^{[i]} = \bar{x}^{[i]} + |S_k^+|d^k$.
        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 (multizoom).
Send the resulting $x^{[i]}$ to the master agent.
```

As it can been seen, DISUD deals with many partitions rather than considering just one partition as it is the case in ISU2D. As a consequence, it is obvious that DISUD is a generalization of ISU2D.

5.1.2 IVD mode algorithm

The IVD idea is to explore near-optimal "LP" neighborhoods incrementally. A worker agent starts by a neighborhood containing potential columns with good reduced costs (computed using the LP duals returned by the master agent) and increments the number of columns as needed. It solves in this neighborhood using Zoom starting with the best solution returned by its own DVD procedure in the competitive case and with the one returned by the master agent in the cooperative mode. Algorithm 3 provides the pseudocode where q' is a parameter tuned by experimentation.

Algorithm 3 IVD pseudocode for agent i

```
Price the columns using $\mu$ (i.e., compute their reduced costs).
Sort the variables in an increasing order of their reduced costs 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 $SPP_k^{[i]}$ by considering the first $k\frac{|J|}{q'}$ variables.
        Solve $SPP_k^{[i]}$ with Zoom, set $\bar{x}^{[i]}$ to the obtained solution, and update $z_{ub}$.
        Send $\bar{x}^{[i]}$ to the master agent.
End for.
```

5.1.3 Illustration

We use the same terminology introduced by Notarstefano and Bullo 2011 to illustrate the mechanics of a worker agent. Like any distributed algorithm, we have:

1. Set of states W: At a time t, a worker agent is either in DVD or IVD mode improving a solution $\bar{x}$ or in an IDLE mode waiting for a message from the master. Thus, the set of the worker agent states W is the set of couples $(mode, \bar{x})$. In addition, we have the END state indicating that the worker agent is done.
2. Messaging function: The set of messages $\mathcal{A}$, called Alphabet, can be subdivided into two subsets. The first one concerns the solution transmission: the worker agent communicates its solution $\bar{x}^{[i]}$ to the master agent and, when worker agents cooperate, the master sends in its turn the best primal solution $\bar{x}_b$. The master also sends the dual solution μ to the worker agents when switching to IVD mode. In order to avoid communication overhead, the sending of a primal solution is accomplished by the transmission of its support only. The second subset contains other messages controlling the state of the worker agent: msgMODE-DVD and msgMODE-IVD to specify which type of decomposition to use, msgMODE-IDLE to stop temporarily a worker agent, msgSTOP to end DISUD.
3. State transition: Upon the reception of a message from the master, a worker agent updates its state as it is illustrated in Figure 1. The latter retraces the state transitions of the agent depending on its current state which would be one of three possible states: IDLE, DVD, or IVD.

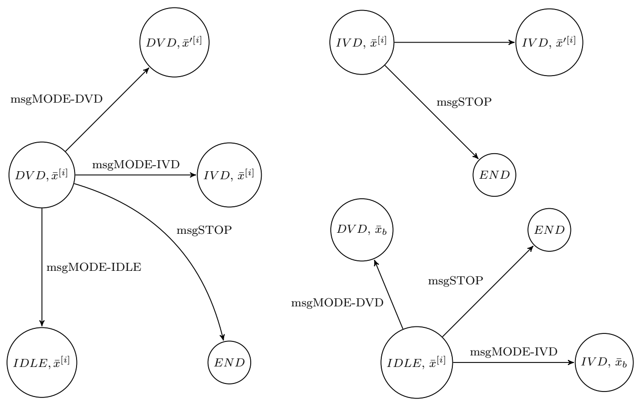

Figure 1: State transition of a worker agent

An illustration of a working agent states transition from the begining of DISUD to the end is given in Figure 2.

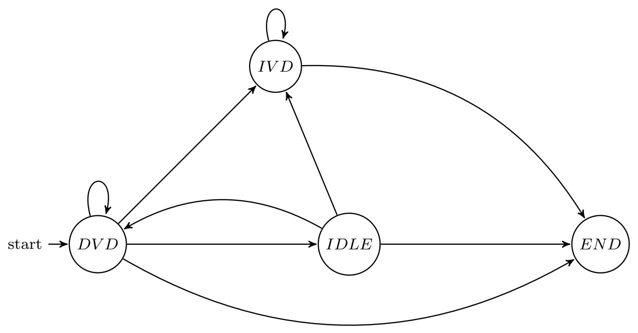

Figure 2: Worker agent basic working

5.2 Master agent

Globally, the master agent calculates a lower bound, controls the execution of DISUD and ends it when a termination criterion is satisfied. We developed two variants of DISUD: 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 5.2.1 and 5.2.2 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 starts with an initial primal solution x_0 of value z_0, sets the upper bound $z_{ub} = z_0$, sends supp(x_0) to all agents, begins to calculate in parallel a lower bound z_{lb} and waits for the solutions obtained by the worker agents. When the master receives a solution $\bar{x}^{[i]}$ from the i^{th} agent that improves the DISUD upper bound z_{ub}, DISUD updates z_{ub} and x_b, the best solution encountered, accordingly. Moreover, if its quality is satisfactory, then DISUD stops the solution process. In the other case, the master reacts according to which variant is activated. The master initializes a counter for each agent $nbrItr^{[i]}$ and increments it after each received solution from the worker agent i. The master agent uses this counter to tell the worker to stay in DVD mode or to move to IVD mode when it reaches a predefined value $IterMax$.

5.2.1 DISUD with cooperative agents

In the cooperative variant, the master intervenes frequently during the progress of the algorithm. This is shown clearly in Algorithm 4. We discuss below the most important issues.

During the DVD mode, the communication between the master and the worker agent is bilateral: the worker agent sends its newly found solution to the master and waits for the solution x_b from which it starts and the OK to stay in DVD mode or any other decision from the master.

The communication is rather unilateral between the master and a worker agent in the IVD mode: the worker agent sends its solution to the master and does not wait for a feedback from the master; the latter just decides to continue or to stop the solution process depending on the quality of the solution it holds. The worker agent does not need x_b. Simply, it improves its $z_{ub}^{[i]}$ until reaching its best solution or get interrupted by the master. To start from different solutions, the master adjusts continously with the maximum number of iterations allowed in DVD mode. When an agent transits to IVD mode, it increment this counter to let other worker agents continue improving x_b in DVD mode.

Finally, if a worker agent does not improve its own solution, the master makes it idle, i.e. waiting for an improved x_b to start from. If all agents are idle, the master let all of them to switch to IVD using different initial solutions.

Algorithm 4 Master cooperative pseudocode

```
Set $z_{ub} = z_0$, $x_b = x_0$ and for each agent $i$, set $nbrItr^{[i]} = 0$, $mode^{[i]} = DVD$
IN-PARALLEL
    Calculate $z_lb$, set $\mu$ to the obtained dual solution
    Send msgSOL, $x_b$, msgMODE-DVD to all worker agents
    Listen to worker agents
    On the reception of $\bar{x}^{[i]}$ from some agent $i$, DO (in sequential)
        Set $nbrItr^{[i]} = nbrItr^{[i]} + 1$
        If $\frac{c^t\bar{x}^{[i]} - z_{lb}}{z_{lb}} \leq \epsilon$, Send msgSTOP to all worker agents; Stop DISUD
        IF $mode^{[i]} = DVD$ THEN
            IF $c^t\bar{x}^{[i]} < z_{ub}$ THEN
                Set $z_{ub} = c^t\bar{x}^{[i]}$, $z_{ub}^{[i]} = c^t\bar{x}^{[i]}$, $x_b = x^{[i]}$
                Send msgSOL, $x_b$, msgMODE-DVD to agent $i$ and idle agents
                and set their mode to DVD
            END IF
            IF $z_{ub} <= c^t\bar{x}^{[i]} < z_{ub}^{[i]}$ THEN
                Set $z_{ub}^{[i]} = c^t\bar{x}^{[i]}$
                IF $nbrItr^{[i]} = IterMax$ OR $\frac{z_{ub} - z_{lb}}{z_{lb}} \leq \epsilon_{dvd}$ THEN
                    Send msgSOL, $x_b$, msgMODE-IVD to $i$
                    Set $mode^{[i]} = IVD$
                    Send $\mu$ to agent $i$
                    Increase $IterMax$ by $\Delta IterMax$ if $nbrItr^{[i]} = IterMax$
                ELSE send msgSOL, $x_b$, msgMODE-DVD to agent $i$
                END IF
            END IF
            IF $c^t\bar{x}^{[i]} >= z_{ub}^{[i]}$ THEN
                Put the agent $i$ in the idle queue and set $mode^{[i]} = IDLE$
                Send msgMODE-IDLE to agent $i$
            END IF
            IF all agents are idle THEN
                Send them msgSOL, $x_b$, msgMODE-IVD, $\mu$
                Change their mode to IVD
            END IF
        END IF
    END DO
IN-PARALLEL
```

5.2.2 DISUD with competitive agents

Concerning the competitive variant, if the received solution improves the i^{th} agent's upper bound $z_{ub}^{[i]}$ only, then the master agent lets the i^{th} agent make another DVD decomposition, of course, if its iteration number does not exceed a predefined value IterMax and the DVD gap threshold ϵ_{dvd} is not reached yet. Otherwise, it sends to the i^{th} agent the message msgMODE-IVD in order to switch to IVD mode. In this case, it sends also the dual values obtained by solving to optimality the linear relaxation. Algorithm 5 presents the master competitive procedure. Note that the instructions comprised between DO and END DO are executed in sequential.

Algorithm 5 Master competitive pseudocode

```
Set $z_{ub} = z_0$, $x_b = x_0$ and for each agent $i$, set $nbrItr^{[i]} = 0$, $mode^{[i]} = DVD$
IN-PARALLEL
    Calculate LP-lower bound for SPP, , set $\mu$ to the obtained dual solution
    Send msgSOL, $x_b$, msgMODE-DVD to all worker agents
    Listen to worker agents
    On the reception of $\bar{x}^{[i]}$ from some agent $i$, DO (in sequential)
        Set $nbrItr^{[i]} = nbrItr^{[i]} + 1$
        IF $c^t\bar{x}^{[i]} < z_{ub}$ THEN
            Set $z_{ub} = c^t\bar{x}^{[i]}$ and $x_b = x^{[i]}$
            IF $\frac{z_{ub}-z_{lb}}{z_{lb}} \leq \epsilon$ THEN; Send msgSTOP to all worker agents; Stop DISUD.
        END IF.
        IF $mode^{[i]} = DVD$ THEN
            IF $c^t\bar{x}^{[i]} = z_{ub}^{[i]}$ OR $nbrItr^{[i]} \geq IterMax$ OR $\frac{z_{ub}-z_{lb}}{z_{lb}} \leq \epsilon_{dvd}$ THEN
                Send msgMODE-IVD to agent $i$ and change its $mode^{[i]}$ to IVD
                Send LP dual values $\mu$ to agent $i$
            ELSE
                Send msgMODE-DVD to agent $i$
                Set $z_{ub}^{[i]} = c^t\bar{x}^{[i]}$
            END IF
        END IF
    END DO
END IN-PARALLEL
```

5.3 Theoretical analysis

We discuss below that working with the partial or the standard reduced costs should give similar results. Let d be an integer descent direction, $S^+ = \{j : d_j > 0\}$, and $P^- = \{l : \lambda_l > 0\}$.

Proposition 1 *Let $\bar{\bar{c}}_j$ the reduced cost of variable j, computed with a dual solution α corresponding to the current integer solution. We have $\sum_{j\in S^+} \bar{\bar{c}}_j = \sum_{j\in S^+} \bar{c}_j$, $\forall\alpha$.*

To prove this, we need the following lemma that can easily be derived from Proposition 9 of Zaghrouti et al. 2014.

Lemma 1 *We have: $v_j = \lambda_l = \dfrac{1}{|S^+|}, \forall(j,l) \in S^+ \times P^-$.*

Proof. (of proposition 1)

Let α be a corresponding dual solution and B the corresponding basis. We have: $\sum_{j\in S^+} A_j = \sum_{l\in P^-} A_l$. So, $\sum_{j\in S^+} c_B^T B^{-1} A_j = \sum_{l\in P^-} c_B^T B^{-1} A_l$. Meaning that $\sum_{j\in S^+}(c_j - \bar{\bar{c}}_j) = \sum_{l\in P^-} c_l$ and consequently $\sum_{j\in S^+}(\bar{\bar{c}}_j) = \sum_{j\in S^+} c_j - \sum_{l\in P^-} c_l$. On the other hand, from Lemma 1, (7), and (11), we obtain $\sum_{j\in S^+} \bar{c}_j = \sum_{j\in S^+} c_j - \sum_{l\in P^-} c_l$. This concludes the proof. □

The next corollary shows a weak "equivalence" between the partial and the standard reduced costs. We think that we do not need a stronger equivalence because theoretically, they behave the same in the worst case.

Corollary 1 *There exist necessarily $j, k \in S^+$ such that $\bar{c}_j < 0$ and $\bar{\bar{c}}_k < 0$.*

We can show that there exists a linear transformation such that $\forall j \in S^+, \bar{\bar{c}}_k < 0$. The next proposition indicates that increasing the number of needed processors (for solving subproblems) beyond a certain limit, that is the diameter of the polytope, is particularly "useless" in DVD mode.

Proposition 2 *Let q_{opt} be the number of CSPs that permit to get an optimal solution x^* in one DVD iteration where each CSP reveals exactly one descent direction. We have $q_{opt} = dist(x^*, \bar{x}) \leq d$ where d is the diameter of $conv(SPP)$.*

Proof. We can show that $supp(x^*) \setminus supp(\bar{x}) = (\cup_{1 \leq k \leq k'} S^+_{k'})$ where S^+_1, S^+_2, ... , $S^+_{k'}$ are minimal disjoint compatible subsets (solutions to the CSP), i.e., their corresponding task subsets T^1, T^2, ..., $T^{k'}$ are disjoint. By definition, the diameter of a polyhedron is the maximum distance between each two of its vertices; note that the distance between two vertices is the minimum number of edges needed to reach the second one, starting from the first vertex. To move on an edge from a vertex to an adjacent one, recall that we need to pivot on a certain S^+_i, a minimal compatible subset of entering columns. Consequently, $q_{opt} \leq k'$. □

Remark 1 *We have two interesting facts:*

- *There exist an infinity of optimal weighting methods we^* such that the weighted graph $G(V, E)$ is a disconnected graph and the resulted complementary subproblems should provide directions leading to an optimal solution x^*.*
- *Let x_1, x_2 be two integer solutions and x_3 and x_4 their respective adjacent extreme points (also integer solutions). We may have $c \cdot x_3 < c \cdot x_4$ even though $c \cdot x_1 >> c \cdot x_2$.*

The next proposition shows the existence of an optimal weighting method which leads to an optimal decomposition.

Proposition 3 *DISUD is a monotonic exact algorithm that converges in a finite time.*

Proof. DISUD is a multi-agent algorithm where every agent converges in a finite time since:

- During DVD phase, the partitions are different and the number of these different partitions is finite.
- During IVD phase, the number of iterations is finite because the number of column subsets is finite.

□

6 Computational results

In this section, we present results of DISUD and discuss the effectiveness of our multi-agent algorithm. We show that DISUD results are interesting compared to DCPLEX.

6.1 Aircrew instances

In aircrew scheduling, a pairing is a sequence of flights that start and end at the same airport. The crew pairing problem CPP consists of finding a set of pairings that covers all the scheduled flights at minimal cost over the planning horizon. Moreover, each flight has to be covered by a single pairing and therefore, CPP is modeled as a SPP.

We tested DISUD on SPP derived from real-life CPP instances. The original datasets can be found in Kasirzadeh et al. 2017 and concern aircraft fleets D94, D95, 757, 319, and 320. We used GENCOL, a commercial software, to generate columns (a set of pairings) using different values for the dominance and incompatibility parameters. We extracted different tests at different phases of the process to build the set of our tests. Therefore, we obtained five groups of instances where each group contains six different instances. Furthermore, considering that DISUD needs an initial solution to start from, we choose a solution from those proposed by GENCOL. Table 1 presents the main characteristics of the instances. It reports for the set of tests the dataset name, the number of tasks m, the minimum number n_{min} and maximum n_{max} of columns before preprocessing. Then it prsents informations of the CPLEX reduced problem obtained after precossing. Indeed, it presents the minimum number m_{min} and maximum m_{max} of tasks and the minimum number n_{min} and maximum n_{max} of columns.

Table 1: main characteristics of instances

	Before preprocessing			After preprocessing			
Serie	Tasks	Variables		Tasks		Variables	
	m	n_{min}	n_{max}	m_{min}	m_{max}	n_{min}	n_{max}
D94	712	58144	110035	453	453	42433	80307
D95	2123	818686	1479515	1340	1344	503046	864673
757	2175	891269	1308428	1422	1423	396889	593987
319	2189	631300	831530	1418	1419	407249	513466
320	2931	689254	1313438	1924	1925	419280	674862

6.2 Testing methodology

We implemented the two variants competitive (DISUD$_{comp}$) and cooperative (DISUD$_{coop}$) using C^{++} and the MPI (Message Passing Interface) library. This latter enables communication between the master and worker agents. The master run on a Linux PC with Quad-Core processor of 3.30 GHz and the worker agents run on Linux PCs with 8 processors of 3.4 GHz each as shown on Figure 3.

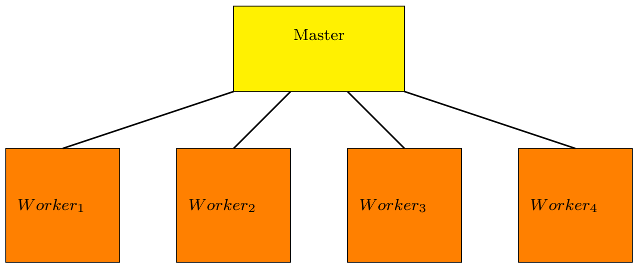

Figure 3: DISUD network architecture

The CPLEX version used here is IBM CPLEX 12.6.1. For each test, algorithms were run within a time limit of one hour except of D94 derived tests which were run during 4 min. DISUD parameters are: $\epsilon_{dvd} = 0.02$, q =4 , IterMax=10 and q'=2. The threshold of 2% comes from industrial observation: a solution with a gap of 1% is considered excellent, and one within 2% is acceptable as reported by Rosat et al. 2017b.

First, we compare DISUD variants. Then we compare these variants to DCPLEX. The latter (see IBM Knowledge Center) is a distributed version of the well known branch and bound algorithm. It is dedicated to solve a MIP in an environment of distributed memory across multiple machines. It is based on a single master associated with multiple workers. DCPLEX presolves the problem on the master and sends the reduced model to each of the workers. Each of the workers then starts to solve the reduced model using its own parameter setting. This phase is known as ramp up. Then, the master selects the worker which performed the best and distributes its search tree over all workers: They work on the same search tree, with the master coordinating the search. We use the following ratios to compare DISUD to DCPLEX:

- The time efficiency ratio $T_a(b)$ between two algorithms a and b is defined as $T_a(b) = 100 * \frac{t(a)}{t(b)}$ where $t(a)$ and $t(b)$ are the computational times of a and b.
- The gap between the found solution value $z(a)$ returned by an algorithm a and the lower bound value z_{lb} is defined as $Gap(a) = 100 * \frac{z(a)-z_{lb}}{z_{lb}}$.

6.3 Cooperative vs Competitive results

In this part, we show DISUD$_{coop}$ and DISUD$_{comp}$ results and discuss their performances on our set of tests. Figure 4 shows the evolution of the objective value over time for DISUD$_{coop}$ on instance D95_1. It presents clearly the contribution of all agents during the process solution. Indeed, it shows the strong point of DISUD: at any moment DISUD solution is the best solution realized by its agents. In addition, we can observe a rapid objective value decrease at the beginning of the solution process while the objective value decrease becomes slow at the end. In other words, the DVD phase (at the begining of the process) yields a large decrease than the IVD phase (at the end). This is explained by the fact that during DVD phase, DISUD combines orthogonal descent directions. This behavior is typical and representative of the other instances.

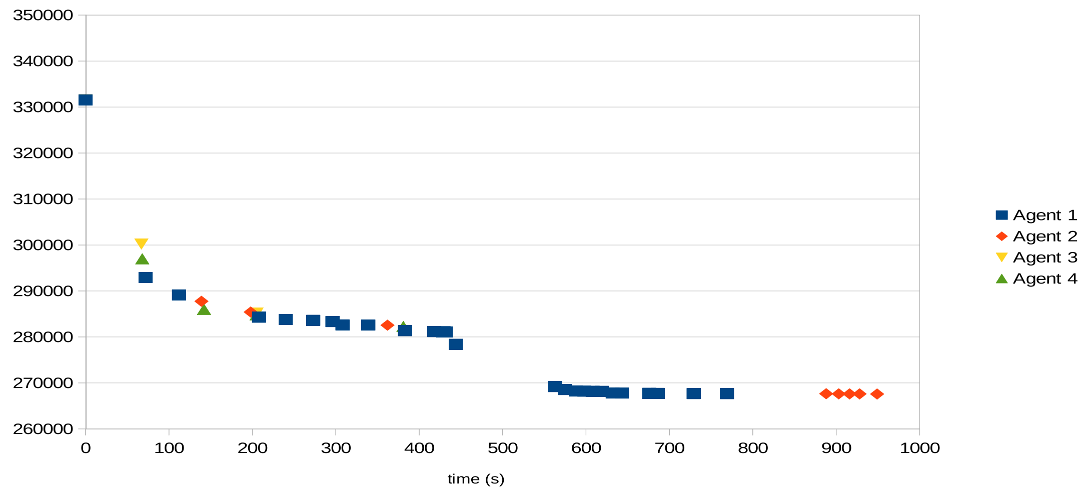

Figure 4: DISUD$_{comp}$ Evolution over time on the D95_1 instance

Figure 5 shows the evolution of DISUD$_{coop}$ and DISUD$_{comp}$ on the test 320_5. We connect the points to improve its readability. We note that the two curves present the same pace. DISUD$_{coop}$ is better in the middle of the process solution. This is due to the fact that DISUD$_{coop}$ embeded the spirit of the branch and bound depth first search strategy. DISUD$_{coop}$ uses all its agents to explore its best solution x_b (solution with the lowest cost) neighborhood.

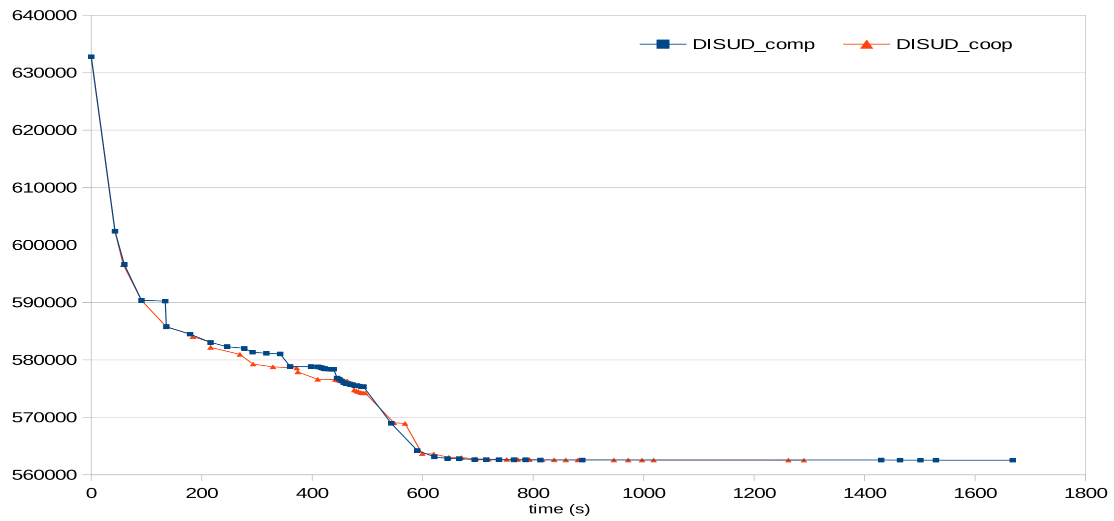

Figure 5: DISUD variants Evolution over time on 320-5 test

The set of tables 2–6 show results for each variant of DISUD. They report test name, number of columns, the gap value of the initial solution, gap values for DVD and IVD phases respectively, the number of times that the best agent invokes a mixed integer program during the SPP resolution and the time devoted to solving these MIP programs, the time to obtain the best solution t_{obj}, for DISUD$_{coop}$ the average time, t_{idle}, that an agent is in idle state and the agent identity Ag_b that reaches the best solution. Also, we have included average lines in bold to compare the average behavior of the two variants.

We observe that both DISUD variants solve all tests to near optimality within almost half an hour. Their solutions quality is less than 1% in all cases. Thus, the results show that the two variants of DISUD were approximately equal in terms of solution quality. Even if the found solutions are excellent, we can see that DISUD$_{comp}$ beats DISUD$_{coop}$ in 37% of the tests in terms of solution quality.

DISUD$_{comp}$ and DISUD$_{coop}$ solved 33% and 64% of tests to less than 2 % during DVD respectively. Furthermore, DISUD$_{coop}$ solved 83% of the 320 and 319 tests to less than 2 % during DVD. These results show that DISUD$_{coop}$ is better than DISUD$_{comp}$ as it is a variant that enables to increase the size of treated problems. Indeed, with large problems, DISUD coop has the potential to use only the DVD decomposition to get industrially acceptable solutions. In addition, the average idle time of an agent is small compared to computing time (less than 1%).

Let us compare DISUD variants computing time as all their solutions are excellent from industrial point of vue. As expected, DISUD$_{comp}$ is faster than DISUD$_{coop}$. The competitive variant beats the cooperative one in 60% of cases. DISUD$_{coop}$ is slower than DISUD$_{comp}$ because it forces worker agents to stay more in DVD phase instead of letting them switch to IVD phase. We would like to point out that the increase of the average DISUD$_{coop}$ computation time differs according to the dataset: it is 0%, 20.9%, 6.7%, 46.9% and -2.1% for D94, D95, 757, 319 and 320 tests respectively.

Concerning Zoom algorithm, we note that MIP influences the performances of DISUD variants: higher the value of MIP time, the lower is the time performance of DISUD. We note that the number of times that a mixed integer program was invoked is small and is approximatively similar for the two variants: they differ in average by one call. Meanwhile, the time reserved to solving these programs differ according to the dataset. Indeed, for DISUD$_{comp}$ the ratio is 1.1%, 35.7%, 8%, 14.2% and 6.8% of the solution process time in average for D94, D95, 757, 319, and 320 instances respectively. For DISUD$_{coop}$ the ratio is 2.1%, 39.9%, 7%, 23.3% and 6.3% of the process solution time in average for D94, D95, 757, 319, and 320 instances respectively. Despite the large proportion of time it could consume, this step has proven to be useful during the resolution: Mip is invoked for all large instances.

In general, DISUD variants perform well when they start with a good initial solution (low initial gap). They reproduce a known fact of primal algorithms which is sensitivity to the initial solution. Their DVD gap decreases with low initial solution gap value.

Finally, it is obvious that all agents contribute to the DISUD process solution as it is shown by the Ag_b column. Based on this, we deduce that considering many agents simultaneously is a better approach. But the agents contributions differ: the first and the second agents are the best for 83% of DISUD$_{comp}$ tests and 66% of DISUD$_{coop}$ tests.

From the aforementioned results, we conclude that DISUD variants yield better results. DISUD$_{coop}$ constitutes a good variant of DISUD that shows a good potentiel to treat larger SPP since its DVD results and the difficulties that may arise when managing the IVD phase. More, DISUD$_{coop}$ allows to manage multiple agents simultaneously in order to take advantage of their actions.

Table 2: DISUD results using D94 instances

			DISUD$_{comp}$						**DISUD$_{coop}$**						
Name	n	**gap**	**gap (%)**		**Mip**		**t$_{obj}$**	**Ag$_b$**	**gap (%)**		**Mip**		**t$_{obj}$**	**t$_{idle}$**	**Ag$_b$**
D94_		Ini	DVD	IVD	nb	**t**	**(s)**		DVD	IVD	nb	**t**	**(s)**	(s)	
D94_1	90637	10.05	4.02	0.15	1	1	23	1	3.84	0.08	0	0	33	2	1
D94_2	110035	5.23	1.55	0.47	0	0	19	2	1.07	0.62	0	0	17	0	4
D94_3	66072	4.50	1.35	0.13	0	0	12	2	0.77	0.17	1	1	10	0	4
D94_4	66952	5.24	1.27	0.40	0	0	7	1	1.27	0.50	0	0	7	0	1
D94_5	70216	10.08	3.15	0.13	0	0	22	2	1.74	0.43	0	0	17	0	1
D94_6	58144	3.00	0.91	0.15	0	0	8	2	0.91	0.14	1	1	9	0	4
Average			**2.04**	**0.24**	**0.17**	**0.17**	**15.17**		**1.60**	**0.32**	**0.33**	**0.33**	**15.50**	**0.33**	

Table 3: DISUD results using D95 instances

			DISUD$_{comp}$						**DISUD**$_{coop}$						
Name	n	**gap**	**gap (%)**		**Mip**		**t**$_{obj}$	**Ag**$_b$	**gap (%)**		**Mip**		**t**$_{obj}$	**t**$_{idle}$	**Ag**$_b$
D95_		Ini	DVD	IVD	nb	**t**	**(s)**		DVD	IVD	nb	**t**	**(s)**	(s)	
D95_1	1060464	24.1	5.85	0.149	7	229	815	1	5.19	0.148	10	92	949	16	2
D95_2	1478737	11.7	3.30	0.166	10	1281	2347	2	3.48	0.157	7	106	1239	51	2
D95_3	898518	24.6	4.07	0.143	7	262	1038	4	3.14	0.150	12	2296	2976	9	1
D95_4	1216846	11.7	3.08	0.140	8	440	1264	4	2.77	0.144	7	43	843	0	1
D95_5	817904	20.3	4.55	0.149	6	44	519	2	3.77	0.148	7	280	986	6	2
D95_6	1144156	9.7	3.14	0.146	7	281	1112	1	3.06	0.169	6	612	1587	0	4
Average			**4.00**	**0.15**	**7.50**	**422.83**	**1182.50**		**3.57**	**0.15**	**8.17**	**571.50**	**1430**	**13.67**	

Table 4: DISUD results using 757 instances

			DISUD$_{comp}$						**DISUD**$_{coop}$						
Name	n	**gap**	**gap (%)**		**Mip**		**t**$_{obj}$	**Ag**$_b$	**gap (%)**		**Mip**		**t**$_{obj}$	**t**$_{idle}$	**Ag**$_b$
757_		Ini	DVD	IVD	nb	**t**	**(s)**		DVD	IVD	nb	**t**	**(s)**	(s)	
757_1	924415	2331.7	2.47	0.008	6	90	868	1	1.87	0.009	4	86	859	0	3
757_2	1271490	2332.3	3.08	0.009	6	67	1105	3	1.88	0.009	4	47	1002	0	1
757_3	1001424	2331.7	1.89	0.008	5	67	900	1	2.22	0.008	6	61	942	6	1
757_4	1307682	2331.8	2.29	0.007	6	75	880	1	1.41	0.005	4	71	993	0	1
757_5	890523	4656.6	2.94	0.009	4	41	786	1	2.44	0.005	8	76	954	0	1
757_6	1139047	2331.8	1.94	0.007	5	79	686	1	1.74	0.006	5	36	825	0	1
Average			**2.44**	**0.01**	**5.33**	**69.83**	**870.83**		**1.93**	**0.01**	**5.17**	**62.83**	**929.17**	**1.00**	

Table 5: DISUD results using 319 instances

			DISUD$_{comp}$						**DISUD**$_{coop}$						
Name	n	**gap**	**gap (%)**		**Mip**		**t**$_{obj}$	**Ag**$_b$	**gap (%)**		**Mip**		**t**$_{obj}$	**t**$_{idle}$	**Ag**$_b$
319_		Ini	DVD	IVD	nb	t	**(s)**		DVD	IVD	nb	t	**(s)**	(s)	
319_1	830774	13.2	2.54	0.106	10	65	671	1	1.97	0.118	11	198	1123	0	2
319_2	786436	2681.9	2.03	0.100	8	101	636	4	1.83	0.099	8	223	819	0	4
319_3	668387	13.3	3.11	0.101	7	93	568	1	2.76	0.099	6	431	1115	8	2
319_4	654470	2682.2	2.16	0.096	7	135	691	2	1.92	0.095	5	100	698	0	4
319_5	630544	13.5	2.80	0.096	7	55	521	1	1.84	0.099	7	155	894	0	3
319_6	638308	8.8	1.84	0.097	7	63	511	1	1.51	0.097	12	124	636	5	2
Average			**2.41**	**0.10**	**7.67**	**85.33**	**599.67**		**1.97**	**0.10**	**8.17**	**205.17**	**880.83**	**2.17**	

Table 6: DISUD results using 320 instances

			DISUD$_{comp}$						**DISUD**$_{coop}$						
Name	n	**gap**	**gap (%)**		**Mip**		**t**$_{obj}$	**Ag**$_b$	**gap (%)**		**Mip**		**t**$_{obj}$	**t**$_{idle}$	**Ag**$_b$
320_		Ini	DVD	IVD	nb	**t**	**(s)**		DVD	IVD	nb	**t**	**(s)**	(s)	
320_1	1077443	11.1	2.94	0.026	5	67	2253	2	1.89	0.026	9	106	1847	0	3
320_2	1312452	3564.8	2.06	0.024	11	87	1919	1	1.92	0.016	8	67	1803	0	1
320_3	862177	12.9	2.55	0.018	10	88	1185	1	2.03	0.020	9	80	1718	0	2
320_4	848669	5.2	1.89	0.015	10	127	1439	2	1.80	0.012	13	202	1541	0	4
320_5	688268	12.5	2.71	0.016	15	234	1668	2	2.37	0.016	10	68	1290	14	2
320_6	791992	3564.5	2.38	0.025	10	75	1466	3	1.85	0.019	16	92	1519	0	1
Average			**2.42**	**0.02**	**10.17**	**113.00**	**1655.00**		**1.98**	**0.02**	**10.83**	**102.50**	**1619.67**	**2.33**	

6.4 Influence of the parameters

In this section, we study the influence of the principal parameters, i.e., of the complementary subproblems number q and the maximum number of iterations during DVD phase. We also give insight into the influence of the initial solution. We present results of the influence of these parameters on the 320_1 test. This choice is directed by the fact that 320 derived instances are sufficiently difficult and DISUD is designed to solve large SPP.

Table 7 gives results for the influence of initial solution throughout its gap value. In general DISUD variants perform well for good initial solutions. Lower the gap of initial solution, the lower is the time of the solution process and the DVD gap. Hence we deduce that DISUD performance increases as initial solution gap decreases.

Table 7: DISUD results using different initial points for 320_1

		MISUD$_{comp}$					**MISUD$_{coop}$**					
Name	**gap**	**gap**		**Mip**		**t$_{obj}$**	**gap**		**Mip**		**t$_{obj}$**	**t$_{idle}$**
320_	Ini	DVD	IVD	nb	**t**	**(s)**	DVD	IVD	nb	**t**	**(s)**	(s)
320_1	1791.9	3.10	0.021	11	154	2347	1.91	0.010	10	225	2014	0
320_1	11.1	2.94	0.025	5	67	2253	1.64	0.011	9	359	2065	0
320_1	8.8	2.10	0.021	10	176	2072	1.91	0.010	8	311	2230	0
320_1	6.5	2.22	0.022	11	110	1783	1.76	0.010	5	184	1860	0
320_1	4.8	1.79	0.019	14	104	1536	1.60	0.009	7	201	1738	0

Table 8 gives results for the influence of the number of iterations, IterMax, during the DVD phase. This parameter controls the duration of the DVD phase. Higher the IterMax value, the higher is the DVD time. We deduce that the DISUD performance increses with the IterMax value. Indeed in general, DVD gap, IVD gap and computing time decrease with the IterMax value. This is explained by the fact that it is likely to get good DVD solution quality as the number of iterations during DVD phase increases.

Table 8: DISUD results using different iterMax for 320_1

		MISUD$_{comp}$						**MISUD$_{coop}$**						
IterMax	**gap$_0$**	**gap$_f$**		**Mip**		**t$_{obj}$**	**Ag$_b$**	**gap$_f$**		**Mip**		**t$_{obj}$**	**t$_{idle}$**	**Ag$_b$**
		DVD	IVD	nb	**t**	**(s)**		DVD	IVD	nb	**t**	**(s)**	(s)	
4	11.1	3.20	0.024	9	66	1147	3	1.75	0.008	6	74	1930	0	3
6		2.96	0.017	10	89	1418	3	1.86	0.009	7	204	1601	5	2
8		2.94	0.020	12	73	1315	2	1.76	0.009	10	143	1543	0	1
10		2.94	0.022	14	77	1266	1	1.88	0.001	11	362	2261	0	3
12		2.94	0.030	9	57	1273	3	1.64	0.007	6	65	1075	0	1

Figure 6 shows the influence of the number of complementary subproblems ,q, on the evolution of DISUD during DVD phase. We choose the values of q so that they are a power of 2 (q =1, 2,4, 8,16). We deduce that we got good performance for both q =4 or 8. This explained by the fact that for lower q (1 and 2) we still solve large complementary subproblems than those obtained with q=4 or 8. For high values (q=16) we got poor performance as it becomes difficult to find descent direction as more variables are ousted by the DVD decomposition process. In addition, there are more processes (16) than processors (8) which leads to the overload phenomen as we used computers with 8 processors.

Therefore, we used the values q =4 , iterMax=10 for the global results given in Sections 6.3 and 6.5

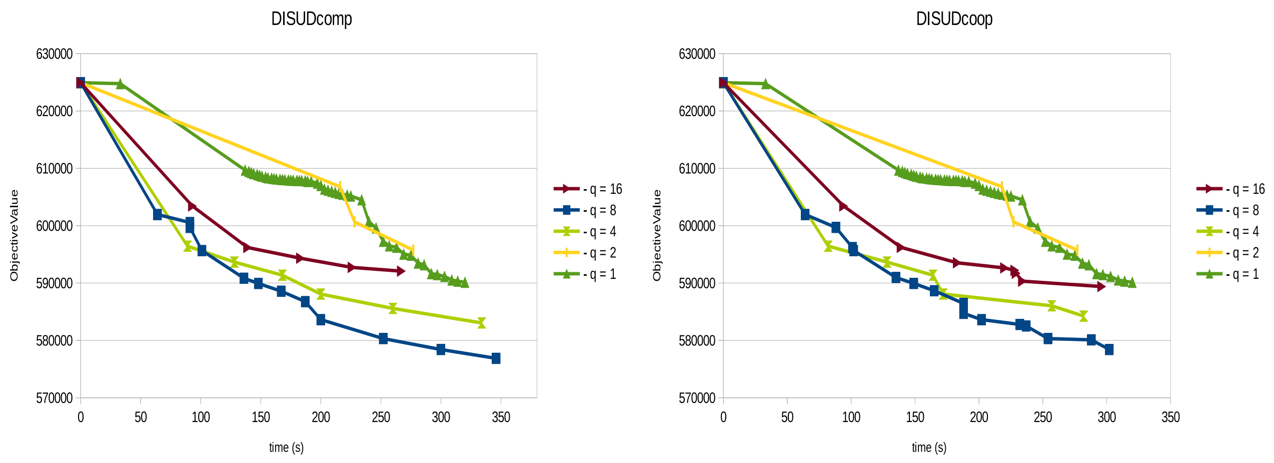

Figure 6: influence of the complementary subproblems number q during DVD phase

6.5 DISUD vs DCPLEX results

The goal in this part is to compare the performances of DISUD$_{coop}$, DISUD$_{comp}$ and DCPLEX using our set of tests. Figure 7 shows the gap value evolution over time for DISUD variants and DCPLEX on the D95_4 test. The figure clearly show that the DISUD variants outperform DCPLEX. The DISUD curves decrease more sharply than the DCPLEX one. DISUD$_{coop}$ is the fastest, yielding improvement of 280% over DCPLEX on this instance. In addition, the number of solutions found through the process resolution by DISUD is great compared to DCPLEX.

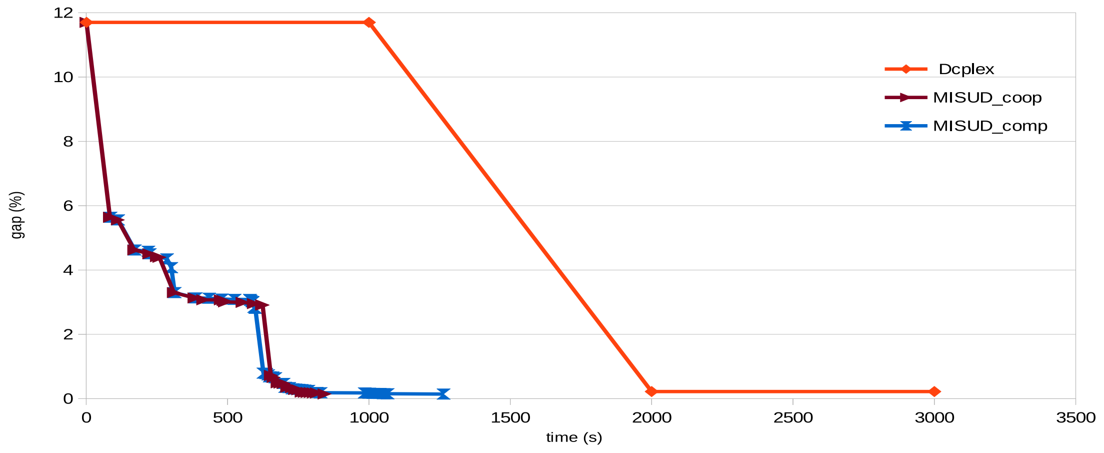

Figure 7: DISUD and DCPLEX Evolution over time on D95_4 instance

The set of tables 9–12 shows DCPLEX results and performance comparison with DISUD variants. They report For DCPLEX, the time to obtain the best solution t_{obj}, the gap value and the number of solution found throughout the solution process. Then for each DISUD variant the time needed to outperform the last DCPLEX objective value: $\mathrm{t}_{cplex}^{DISUD}$, the number of solution found during the resolution and improvement realized.

For D94 derived tests, we observe that DCPLEX was able to get good quality solution in less time than did DISUD.

For D95, 757,319 and 320 derived tests, we observe that DCPLEX was unable to improve the initial solution for 40% of instances within a time limit of an hour whereas bothDISUD variants solve 87% instances to near optimality (less than 1%) within half an hour and all instances within Three quarters of an hour. Furtehrmore DCPLEX was unable to improve any test of the 757 serie and 50% of the 319 derived tests.

We note also that DISUD algorithm find large number of different integer solutions than DCPLEX throughout the solution process: the ratio is between 5 and 19 times. We mention that this property is appreciable when solving the optimization problems because it permits to get an overview of the resolution process and to stop it once a satisfactory solution is found.

We conclude that based on our tests, DISUD outperforms DCPLEX in terms of the solution quality and the resolution time for large instances. On the other hand, DCPLEX is more efficient on small instances.

Table 9: DCPLEX and DISUD comparison on D94 instances

Instance		DCPLEX			DISUD$_{comp}$				DISUD$_{coop}$			
Name	gap$_0$ (%)	t$_{obj}$ (s)	gap$_f$ (%)	nSol (%)	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp (%)
D94_1	10.05	7	0.08	7	0.15	-	20	-	0.08	-	15	-
D94_2	5.23	13	0.10	2	0.47	-	11	-	0.62	-	12	-
D94_3	4.50	5	0.05	4	0.13	-	11	-	0.17	-	4	-
D94_4	5.24	7	0.09	9	0.40	-	9	-	0.50	-	5	-
D94_5	10.08	8	0.07	12	0.13	-	17	-	0.43	-	9	-
D94_6	3.00	5	0.06	5	0.15	-	14	-	0.14	-	8	-

Table 10: DCPLEX and DISUD comparison on D95 instances

Instance		DCPLEX			DISUD$_{comp}$				DISUD$_{coop}$			
Name	gap$_0$ (%)	t$_{obj}$ (s)	gap$_f$ (%)	nSol (%)	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp (%)
D95_1	24.1	2600	0.20	6	0.149	623	34	417.34	0.148	686	40	379.01
D95_2	11.7	2890	0.27	5	0.166	1024	33	282.33	0.157	793	35	364.44
D95_3	24.6	2500	0.27	5	0.143	478	42	523.01	0.150	719	52	347.71
D95_4	11.7	2000	0.22	3	0.140	704	44	284.09	0.144	711	32	281.29
D95_5	20.3	1856	0.22	7	0.149	331	30	560.73	0.148	673	41	275.78
D95_6	9.7	3234	0.41	10	0.146	896	52	360.94	0.169	905	36	357.35

Table 11: DCPLEX and DISUD comparison on 319 instances

Instance		DCPLEX			DISUD$_{comp}$				DISUD$_{coop}$			
Name	gap$_0$ (%)	t$_{obj}$ (s)	gap$_f$ (%)	nSol (%)	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp (%)
319_1	13.2	3019	0.17	6	0.106	550	35	548.91	0.118	678	55	445.28
319_2	2681.9	3247	0.20	5	0.100	421	22	771.26	0.099	544	57	596.88
319_3	13.3	3600	-	-	0.101	-	30	-	0.099	-	58	-
319_4	2682.2	3600	-	-	0.096	-	35	-	0.095	-	38	-
319_5	13.5	2030	0.25	4	0.096	312	48	650.64	0.099	437	51	464.53
319_6	8.8	3600	-	-	0.097	-	38	-	0.097	-	48	-

Table 12: DCPLEX and DISUD comparison on 320 instances

Instance		DCPLEX			DISUD$_{comp}$				DISUD$_{coop}$			
Name	gap$_0$ (%)	t$_{obj}$ (s)	gap$_f$ (%)	nSol (%)	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp	gap$_f$ (%)	t$_{cplex}^{DISUD}$ (s)	nSol	Imp (%)
320_1	11.1	3600	-	-	0.026	-	43	-	0.026	-	67	-
320_2	3564.8	3243	1.534	5	0.024	1010	68	321	0.016	915	55	354
320_3	12.9	1887	0.025	6	0.018	1018	37	185.36	0.020	1408	66	134.02
320_4	5.2	2387	0.018	6	0.015	1305	54	182.91	0.012	1271	44	187.80
320_5	12.5	2323	0.016	6	0.016	1668	57	139.27	0.016	1290	54	180.07
320_6	3564	2053	0.036	10	0.025	1402	24	146.43	0.019	1170	71	175.47

7 Conclusion

We proposed a distributed version of ISUD. It is a multi-agent based algorithm dedicated to find descent directions leading to improved integer solutions. We presented and implemented two DISUD variants and discussed their performance. They differ in the strategy used to manage network agents. We showed that our algorithm yields better results than the distributed version of CPLEX on a set of instances derived of industrial aircrew scheduling. Our tests set contains large-scale instances with up to almost 2,000 flights and 1,300,000 pairings. They are given as set-partitioning problems associated with initial solutions. We demonstrated that the cooperative strategy gives good results and shows good potentiel. DISUD was able to find near optimal solutions for all large instances in less time than that required by DCPLEX. DISUD realized a time efficiency ratio between 150% and 770%. More, DCPLEX is unable to produce solutions as good as those that DISUD produce within the same time limit.

Future research on network agents management strategies should be done to further improve the DISUD performance. In addition, other agents could be added also to the network. In addition, combining DISUD with heuristics that produce good initial solutions should significantly lead to obtain good DISUD performances.

References

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.

M. Bürger, G. Notarstefano, F. Bullo, and F. Allgöwer. Distributed abstract optimization via constraints consensus: theory and applications. Automatica, 48(1):2298–2304, 2012.

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.

J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming. Thinking Machines Corporation Technical Report TMC, 257, 1993.

I. El Hallaoui, A. Metrane, F. Soumis, and G. Desaulniers. An improved primal simplex algorithm for degenerate linear programs. INFORMS Journal on Computing, 23(4):569-577, 2011. doi: http://dx.doi.org/10.1287/ijoc.1100.0425.

M. Fischetti, M. Monaci, and D. Salvagnin. Selfsplit parallelization for mixed-integer linear programming. Computers and O.R. (93), 101–112, 2018.

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

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.

B. Gendron and T. Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. Operations Research, 42:1042–1066, 1994.

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

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.

P. Laursen. Simple approaches to parallel branch and bound. Parallel Computing, 19:143–152, 1993.

A. Letchford and A. Lodi. Primal cutting plane algorithms revisited. Math. Methods Oper. Res. 56(1):67–81, 2002.

G. Notarstefano and F. Bullo. Distributed abstract optimization via constraints consensus: theory and applications. IEEE Transactions on Automatic Control, 56(10):2247–2261, 2011.

M. Quinn. Analysis and implementation of branch and bound algorithms on a hypercube multicomputer. IEEE Transactions on Computers, 39:384–387, 1990.

T. Ralphs, Y. Shinano, T. Berthold, and T. Koch. Parallel solvers for mixed integer linear optimization. COR@L Technical Report 16T-014-R3 (ISE, Lehigh University) and Zuse Institute Berlin (ZIB) Technical Report 16–74, 2017.

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

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, D. Chakour, and A. Lodi. Integral simplex using decomposition with primal cutting planes. Mathematical Programming, doi:10.1007/s10107-017-1123-x., 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, doi.org/10.1016/j.ejor.2017.05.047., 263:1007–1018, 2017b.

A. Zaghrouti, F. Soumis, and I. El Hallaoui. Improving ilp solutions by zooming around an improving direction. cahiers. Cahiers du Gerad, G–2013–107, HEC Montreal, 2013.

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