Les Cahiers du GERAD ISSN: 0711–2440

Integral simplex algorithm disaggregating tasks for the generalized set partitioning problem

A.-S. Barry, F. Quesnel, I. El Hallaoui, F. Soumis

G–2026–74

October 2026

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

Citation suggérée : A.-S. Barry, F. Quesnel, I. El Hallaoui, F. Soumis (Octobre 2026). Integral simplex algorithm disaggregating tasks for the generalized set partitioning problem, Rapport technique, Les Cahiers du GERAD G– 2026–74, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2026-74) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Suggested citation: A.-S. Barry, F. Quesnel, I. El Hallaoui, F. Soumis (October 2026). Integral simplex algorithm disaggregating tasks for the generalized set partitioning problem, Technical report, Les Cahiers du GERAD G–2026–74, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2026-74) to update your reference data, if it has been published in a scientific journal.

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, 2026
– Bibliothèque et Archives Canada, 2026

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, 2026
– Library and Archives Canada, 2026

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Integral simplex algorithm disaggregating tasks for the generalized set partitioning problem

Alpha-Saliou Barry [a, c]

Frédéric Quesnel [b, c]

Issmail El Hallaoui [a, c]

François Soumis [a, c]

[a] *Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4*

[b] *Département d'analytique, opérations et technologies de l'information, Université du Québec à Montréal, Montréal (Qc), Canada, H2X 1L7*

[c] *GERAD, Montréal (Qc), Canada, H3T 1J4*

alpha-2-saliou.barry@polymtl.ca
quesnel.frederic@uqam.ca
issmail.elhallaoui@gerad.ca

October 2026
Les Cahiers du GERAD
G–2026–74

Copyright © 2026 Barry, Quesnel, El Hallaoui, Soumis

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract : The set partitioning problem (SPP) consists in finding a partition of a set of tasks with minimum cost. It is a well-studied problem for which efficient solution methods exist. A generalization of SPP is to find a set of parts, still with minimal cost, that covers each task a given number of times (number fixed in advance). This generalization no longer possesses the quasi-integrality property. In this paper, we propose a new algorithm based on a task disaggregation strategy which, by splitting tasks into several subtasks, brings the generalization close to the classical set partitioning problem (and then close to the quasi-integrality property). We then show, experimentally, that our new algorithm is more than 5 times faster than a state-of-the-art branch-and-cut algorithm.

1 Introduction

In this article, we study a generalization of the set partitioning problem where the right-hand side is composed of small positive integers. This context has many applications in the airline industry - which notably triggered this study, in freight transportation (split delivery), and in rail transportation (locomotive assignment), to name a few. The mathematical formulation of the generalized set partitioning problem (GSPP) is given below:

$$\text{GSPP} = min\ c^T x \tag{1}$$

$$s.t.$$

$$Ax = b \tag{2}$$

$$x \in \{0,1\}^n \tag{3}$$

where n represents the number of variables, A is a $m \times n$ binary constraint matrix and b is a right-hand side of size m containing only strictly positive integers. When b_i ¿ 1, for $i \in [\![1, m]\!]$, the task i has to be performed several times. For instance, a flight leg should be covered by a certain number of crew members, a customer must be delivered by a given number of trucks in case of split delivery, and so on. When relaxing (the integrality of) variables to be in the interval [0, 1], we obtain the linear relaxation denoted by GSPP_r.

GSPP is a generalization of the set partitioning problem (SPP) which corresponds to the special case where $b_i = 1\ \forall i \in [\![1, m]\!]$. Here again, SPP_r refers to the linear relaxation of SPP. SPP has extensively been studied in the past five decades. One family of methods that has recently had good success in solving it in practice is integral simplex algorithms. An integral simplex algorithm starts with an initial integer solution and iteratively computes adjacent integer solutions of decreasing cost until it reaches optimality. These algorithms exploit the quasi-integrality property of SPP_r: every edge of the convex hull of SPP integer points is also an edge of SPP_r. A current solution $\bar{x}$ can then be improved by performing simple pivots of the simplex algorithm.

In Barry et al. (2024), we observed that, unlike SPP, GSPP no longer possesses the quasi-integrality property: it is possible to have in practice GSPPs where some integer solutions are isolated, i.e., they do not have integer neighbors in the polytope of GSPP_r. Consequently, we cannot directly use integral simplex type algorithms. We propose here a new method to recover the quasi-integrality property. The proposed method dynamically disaggregates certain tasks, which has the effect of decreasing b_i at the expense of an increase in the number of variables and constraints. Disaggregation simply consists in creating task copies that are covered fewer times. The disaggregated problem gets closer to SPP and thus better suited for an integral simplex algorithm. In particular, a solution in the disaggregated problem is more likely to have an adjacent integer neighbor of lower cost. Indeed, when the GSPP is fully disaggregated, all right-hand sides are equal to 1. The GSPP then corresponds to an SPP, which has the property of quasi-integrality. Section 4.2 shows, however, that the problem size can explode rapidly. The level of disaggregation must be kept low in order to retain the problem size tractable. Ideas to judiciously disaggregate the problem are discussed theoretically, algorithmically and numerically. We show that in many cases, the new method improves the current solution with no branching. This fact helps reduce the average computation time by a factor close to 5 on some real instances of the aircrew pairing problem.

The remainder of this paper is organized as follows. Section 2 presents a literature review. Section 3 presents ISUD and GISUD algorithms. Section 4 presents task disaggregation strategy and Section 5 presents the new algorithm implementing the proposed strategy. Section 6 presents the results and finally, Section 7 concludes the article.

2 Literature review

We focus in this paper as well as in the previous one (Barry et al. (2024)) on GSPP as a distinct class of problems. Unfortunately, there is little literature on GSPP. It has always been dealt with as a general integer program (IP) and thus solved by a general *branch and cut* method (Land and Doig, 1960). Branch and cut is a dual method, as it essentially improves a lower bound (in the case of minimization) with branching or cutting. The method stops when this lower bound equals the upper bound (i.e., the cost of the best integer solution found so far). Generally, only a few integer solutions are produced during the optimization process. Even worse, it is often necessary to wait until the end of the optimization process to obtain an integer solution. This represents a major issue for large-scale industrial problems. To overcome this issue, we witnessed recently an increased use of primal heuristics within commercial solvers like CPLEX and Gurobi. This certainly improved the solvers, but the issue has not been completely resolved and cannot in any case be considered a primal algorithm, which is the main topic of the paper. A primal (integral) algorithm is by definition an algorithm that computes a sequence of integer solutions. If each two consecutive integer solutions are necessarily adjacent, we call them integral simplex algorithms. If the final solution of the sequence is guaranteed to be optimal, we have an exact primal algorithm. Otherwise, they are heuristic. One of the strengths of primal algorithms is the fact that when the solution process stops prematurely (due to time limits for instance), we have an integer solution at hand, which is highly desired in practice.

2.1 Heuristic primal algorithms

Metaheuristics are widely used in practice to solve combinatorial problems because they have the advantage of being particularly fast, easily implementable, and more importantly because they are primal. We generally start with an integer solution (or a population of them (Lambora et al., 2019)) and improve it intensively through local search according to certain criteria until a local optimum (Tabu Search, Glover (1989) for instance). To escape such local optima, we resort to diversification strategies. Typical strategies involve defining multiple neighborhoods (Variable Neighborhood Search, Mladenović and Hansen (1997)) or doing multi-starts (GRASP, Feo and Resende (1995)). The solution sequence(s) produced by metaheuristics are generally non-monotonic. In these metaheuristics, the integral feasibility of the solutions is kept invariant, and one or more sequences of integral solutions are computed. The literature on metaheuristics is abundant (Gendreau and Potvin, 2005).

However, these methods are not generic enough (the operators, neighborhood definitions, ...) and generally need fine-tuning each time we have a new problem, sometimes even when we have a new instance of the same problem. Furthermore, they are not exact and in most cases, we lack guarantees on the quality of the solution we find, especially in highly constrained large-scale environments. This is problematic in contexts where 1% might represent millions of dollars. The airline industry is a good example of such contexts.

2.2 Exact primal algorithms

Finding a primal exact algorithm for integer programming has been a dream of the operations research community since the 1960s. Very renowned researchers worked on that in the past for its importance. There is, for example, the (Ben-Israel and Charnes, 1962) algorithm, which starts with an integer solution, performs pivots leading to an improved solution, adds cutting planes if the pivot leads to a fractional solution, and so on. Many variants of these algorithms have been developed (Young, 1965, 1968; Glover, 1968; Letchford and Lodi, 2002).

Recently, Geometric Scaling (Deza et al., 2024) has been proposed in order to reduce the number of iterations for general binary problems. This is an oracle-based theoretical method with a pseudo-polynomial number of calls to the oracle (iterations, in the order of $O(nlog(n)||c||_\infty)$ in the worst case), which is very interesting from a theoretical point of view. The method assumes hypothetically that

we have an efficient oracle. The difficulty, of course, lies in finding such an oracle. Even though these methods are either theoretical or time-consuming, which limits their implementability in practice, we consider them as foundational works. Our recent works on integral simplex algorithms (next section) fill some gap in the literature by proposing an oracle that works in practice and thus answers a key open question.

2.3 Integral simplex algorithms

Balas and Padberg (1972) proved that from every non-optimal integer point x of the SPP, there exists a better-cost adjacent integer solution in SPP_r. Consequently, they propose an algorithm that enumerates the neighbors (adjacent bases) of an integer point in SPP_r until a better-cost integer neighbor is found. There is also (Thompson, 2002)'s algorithm, which is an all-integer algorithm that uses a branching strategy when stuck in a local optimum. At each node of the branching tree, an integer solution is improved by pivoting only on a 1 coefficient. These pivots preserve the integral feasibility of the solution. These enumerative methods are time-consuming in practice, as there is often an exponential number of adjacent bases associated with an extreme point, degenerate by essence. The branching tree is also exponential.

The integral simplex using decomposition (ISUD) algorithm resolves this degeneracy issue. It is currently one of the most promising algorithms for solving SPP. ISUD attempts to find a better solution that is adjacent to the current one by calling an oracle consisting simply in solving a reduced (linear) problem (RP) or its complementary (CP). Solving these simple linear programs often yields an improving integer solution. If this search fails, a branching in the CP is performed. Subsequent improvements to ISUD have been proposed. One of them is to add cutting planes to the CP (Rosat et al., 2017) instead of branching, or to replace branching by a succession of small IPs called a ZOOM procedure (Zaghrouti et al., 2020). In cases where not all variables are enumerated in advance, it is also possible to combine ISUD with column generation by solving the restricted master problem with ISUD, generating new columns using the subproblem(s) (Tahir et al., 2019), and repeating until satisfaction.

Unfortunately, integral simplex algorithms developed for SPP cannot be used directly for GSPP. This is mainly due to the lack of the quasi-integrality property as mentioned earlier. We developed in Barry et al. (2024) a column addition strategy within ISUD to recover this property, at least locally, by increasing the dimension of GSPP, i.e., adding new columns designed specifically for creating the needed adjacency. ISUD and GISUD, its generalization to GSPP, are as practical as heuristic primal algorithms, with the advantage of being exact and monotonic. The next section provides more details on ISUD and GISUD.

In this paper, we propose a new strategy for recovering quasi-integrality in GSPP_r. This new strategy, called *task disaggregation*, leads to a new algorithm: GISUD^{dis}, which can be used in standalone mode or hybridized with the column addition strategy.

3 ISUD and GISUD algorithms

Let $\bar{x}$ be a solution to (G)SPP, $\mathcal{P} = \{j \in [\![1, n]\!] : \bar{x}_j = 1\}$, and $\mathcal{Z} = \{j \in [\![1, n]\!] : \bar{x}_j = 0\}$. Observe that $\mathcal{P} \cup \mathcal{Z} = [\![1, n]\!]$. This representation of $\bar{x}$ as $(\mathcal{P}, \mathcal{Z})$ has the particularity of being unique, unlike representing $\bar{x}$ as a simplex basis, which is meaningless in the presence of degeneracy. In this article, variables and their associated columns are used interchangeably to mean the same thing.

ISUD/GISUD first tries improvements that are easier to achieve by solving RP. At each iteration, RP identifies a variable of $\mathcal{Z}$ with negative reduced cost, if any, whose value is increased to 1 and, one or more variables of $\mathcal{P}$ whose value is decreased to 0. Here, we keep the solution feasible. This strictly improves the objective value. Such variables are said to be *compatible/binary compatible* in the

ISUD/GISUD jargon. RP consequently updates $\bar{x}$, $\mathcal{P}$, and $\mathcal{Z}$ and reiterates until there are no more such improvements.

Once RP can no longer find better integer solutions, CP is solved to find an improving direction d. The special feature of CP lies in its simplicity: it is a linear program that is easy to solve. CP returns the normalized direction (the set of feasible directions heading from $\bar{x}$ is a cone) with the lowest reduced cost. For a set of indices $\mathcal{X}$, $A_{\mathcal{X}}$ corresponds to the matrix A restricted to the columns indexed by $\mathcal{X}$, and similarly for other vectors and matrices. Problem (4) – (8) corresponds to the CP associated with the solution $\bar{x} = (\mathcal{P}, \mathcal{Z})$.

$$CP(\mathcal{P}, \mathcal{Z}) = \min c_{\mathcal{P}}^T d_{\mathcal{P}} + c_{\mathcal{Z}}^T d_{\mathcal{Z}} \tag{4}$$

$$s.t.$$

$$A_{\mathcal{P}} d_{\mathcal{P}} + A_{\mathcal{Z}} d_{\mathcal{Z}} = 0 \tag{5}$$

$$-w_{\mathcal{P}}^T d_{\mathcal{P}} + w_{\mathcal{Z}}^T d_{\mathcal{Z}} = 1 \tag{6}$$

$$d_{\mathcal{P}} \leq 0 \tag{7}$$

$$d_{\mathcal{Z}} \geq 0 \tag{8}$$

Constraints (5), (7), (8), called compatibility constraints, ensure the feasibility of the direction $d = (d_{\mathcal{P}}, d_{\mathcal{Z}})$ and constraint (6) bounds the cone of such feasible directions. To do so, the weights $w = (w_{\mathcal{P}}, w_{\mathcal{Z}})$ should be strictly positive.

When $\bar{x}$ is not optimal, solving CP yields a feasible direction of negative reduced cost. It is shown in the ISUD (Zaghrouti et al., 2014) and GISUD (Barry et al., 2024) that the direction returned by CP always corresponds to a move along an edge of SPP_r or GSPP_r, adjacent to $\bar{x}$. Since the SPP has the quasi-integrality property, there is always a possibility that CP returns an integral direction. Unfortunately, $GSPP$ does not have this property. Consequently, the solution of CP from such a point may never return an integral direction, as $\bar{x}$'s neighbors may be all fractional.

In many applications (crew pairing or vehicle routing problems), d frequently leads to an improved integer point, and is thus called an integral direction. If CP returns an integral direction, a new iteration starts from the new integer point obtained with the direction. Otherwise, if CP returns a fractional direction, a $ZOOM$ procedure is called to ensure convergence of the algorithm. The ZOOM procedure solves a succession of integer problems of increasing size in order to find a better integer solution or to prove optimality. The ZOOM procedure always succeeds in finding a better integer solution if there is one. A new iteration ends when a new solution is found by either RP, CP or $ZOOM$.

To speed up the solution process, a multi-phase strategy is used in CP. Columns are first ordered according to their *incompatibility degree*: a certain distance measure to the span of the columns in $\mathcal{P}$, i.e., the current solution. Compatible columns/binary compatible columns therefore have an incompatibility degree equal to zero. We draw attention here to the fact that RP contains only (binary) compatible columns. The multi-phase strategy first solves RP, then CP by considering only columns of $\mathcal{Z}$ with an incompatibility degree lower than 1. In the case of failure, the phase (i.e., the maximum incompatibility degree tolerated in CP) increases until the direction returned by CP has a negative reduced cost, provided $\bar{x}$ is not yet optimal. Several experiments (Zaghrouti et al., 2014) have shown that early phases are very quick and favor integrality. A sophisticated procedure for computing the incompatibility degree is provided in Section 3.1.

3.1 Particularity of the GISUD algorithm

Barry et al. (2024) propose a column addition strategy that builds on a fractionnal CP direction and attempts to complete it to form an integer direction. Ths CP solution d is aggregated into a single artificial column and is pivoted into $\mathcal{P}$, thus creating an artificial solution. The CP is solved from

that solution in an attempt to find an integral direction that includes d. Barry et al. (2024) provides a mathematical justification and experimental validation for this strategy. However, column addition can only be attempted if the CP solution satisfies some property, thus limiting its usefulness.

Another particularity of GISUD is the means of computing the incompatibility degree of a column. The incompatibility degree of a column allows us to measure the column's proximity to the current solution. The lower the incompatibility degree, the more "compatible" the column is with the current solution. Restricting the columns of the CP problem to columns with a low incompatibility degree has the effect of favoring integer directions. The columns most compatible with the current solution are binary compatible columns, which have an incompatibility degree of zero. It is clear that pivoting on such a column preserves the integrality of the solution.

We denote by W_k the tasks covered by column A_k, the column for which we want to compute the incompatibility degree, and by W_j the set of tasks covered by column $A_j, j \in \mathcal{P}$. An assignment f corresponds to a function that associates to each task $w \in W_k$ a column of $\mathcal{P}$ that covers the task w (if $f(w) = j$ then $w \in W_j$). Each assignment corresponds to a slicing of some of the columns of $\mathcal{P}$, which allows column A_k to be written as a sum of slices of the columns of $\mathcal{P}$ (then A_k becomes binary compatible with $\mathcal{P}$ slices). In fact, for each column $A_j, j \in \mathcal{P}$, $f^{-1}[\{j\}]$ corresponds to a subset of W_j tasks (possibly empty). The column A_j is sliced into two if $f^{-1}[\{j\}] \neq \emptyset$ and $f^{-1}[\{j\}] \neq W_j$. These two sub-columns cover the tasks $f^{-1}[\{j\}]$ for the first slice and the remaining tasks in W_j ($W_j \backslash f^{-1}[\{j\}]$) for the second slice. This slicing then allows column A_k to be written as a sum of column slices of the columns of P (after slicing). The incompatibility degree of column A_k corresponds to the minimum number of columns of $\mathcal{P}$ that must be sliced in two in order to make column A_k binary compatible. The number of sliced columns for assignment f is exactly: $\eta(f) = \sum_{j \in \mathcal{P}} \mathbb{1}(f^{-1}[\{j\}] \neq \emptyset \wedge f^{-1}[\{j\}] \neq W_j)$. The incompatibility degree of column k is then :

$$\min_f \eta(f)$$

Since determining the incompatibility degree is a combinatorial problem and could be hard to solve, we prefer to approximate it by solving a shortest-path problem. We construct a graph in which each path is associated with an assignment (a way of slicing the columns of $\mathcal{P}$ so that A_k is written as a sum of slices of $\mathcal{P}$ columns). The length of a path corresponds to the number of columns of $\mathcal{P}$ that the corresponding assignment would slice. We then solve a shortest-path problem to obtain the assignment that slices the fewest columns. The tasks of $[\![1, m]\!]$ are ordered in a predetermined sequence (e.g. the start time of each task). For column $A_j, j \in \mathcal{P}$ and for task $i \in W_j$, we denote by $pred(i, j)$ the task of W_j that precedes i according to the established order. Similarly, we denote by $succ(i, j)$ the task of W_j that follows i. These values are "NIL" if there is no predecessor or successor to task i in W_j.

The nodes of the graph are arranged in stages from top to bottom, and each stage has one or more nodes corresponding to the same W_k task in sequence. The first stage corresponds to the first task of W_k in the task order. For a stage corresponding to a task $i \in W_k$, there is a node (w_i, A_j) for each column $j \in \mathcal{P}$ covering task i. There is a directed arc between each node of stage i and each node of stage $succ(i, k)$.

A source is added before the first stage and is connected to all nodes on the second stage, and likewise, a sink is added after the last stage and is connected to all nodes on the last stage. Each path in the graph linking the source to the sink corresponds to an assignment, since each stage is associated with a W_k task, and each path passes through a single column of $\mathcal{P}$ on each stage: on a path, each W_k task is therefore associated with a column of $\mathcal{P}$ covering the task. Each arc is assigned a length: 0, 1, or 2, according to the algorithm in Appendix A, which computes the length L of the arc between node (w_i, A_j) and node $(w_{i'}, A_{j'})$. The length L is chosen such that the length of a path in the incompatibility degree graph is an upper bound on the size of the corresponding assignment. The shortest path between source and sink then gives an upper bound on a column's incompatibility measure. The following example illustrates the procedure to compute the incompatibility degree:

The tasks are in the order 1, 2, 3.

$$A = \left(\begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}\right) \text{ and } b = \begin{pmatrix}1\\2\\2\end{pmatrix}$$

Suppose $\mathcal{P}$ corresponds to columns $\{1, 2, 3\}$ of A and that $k = 4$. We want to compute the incompatibility degree of column A_4 with respect to $\mathcal{P}$. To do this, we construct the graph using the procedure described above:

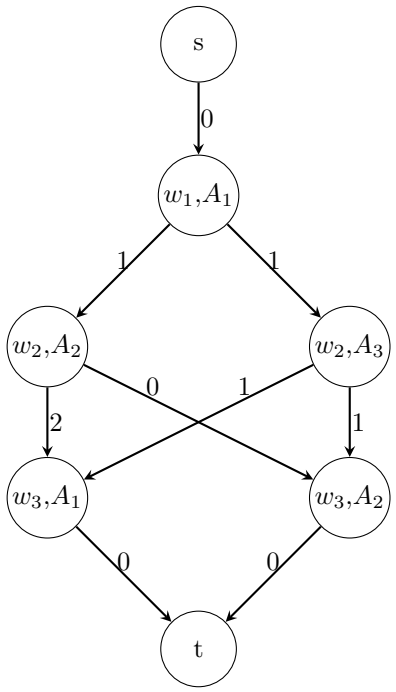

Figure 1: Oriented graph corresponding to the example

The shortest path s–(w_1,A_1)–(w_2,A_2)–(w_3,A_2)–t is of length 1 in this graph, so the approximation of the incompatibility degree of column A_4 is 1. The true value of the incompatibility degree is actually 0, as column A_4 is written as the sum of columns A_1 and A_3.

4 Task disaggregation

As mentioned above, the probability of success of the CP is closely correlated with how close the GSPP is to having the quasi-integrality property. The main idea behind the contribution of this paper is to partially restore this property by disaggregating some tasks (i.e. considering a task that must be covered several times as several tasks that must be covered a single time). Doing so reformulates the GSPP in a way that more resembles the SPP. This section formalizes task disaggregation.

In Section 4.1, the precise definition of a task disaggregation is first presented. We then show how to obtain an equivalent GSPP problem after disaggregation in Section 4.2. This disaggregated problem has lower right-hand side values. For ease of reading, we consider the following GSPP example throughout the section:

Example 1

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

Note that the column costs are not specified as they are not relevant for our demonstrations.

4.1 Task disaggregation definition

In Example 1, all three tasks have an associated right-hand side equal to 2. Suppose we want to "disaggregate" task 1 into two sub-tasks, both of which have right-hand sides equal to 1. The right-hand side then becomes $\tilde{b} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \end{pmatrix}$ and contains 4 tasks instead of 3. By numbering the new tasks, in order, 1, 2, 3 and 4, the disaggregation can be represented by the Figure 2, where the nodes on the left represent the tasks of the initial problem and the nodes on the right represent the tasks after disaggregation. This process can be generalized and encoded in vector form. A task disaggregation then consists of defining a new set of tasks, an association of the new tasks with the old tasks, and a right-hand side associated with the new tasks. The formal definition of a disaggregation is shown below:

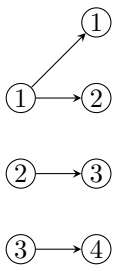

Figure 2: Representation of task disaggregation

Definition 1 (Disaggregation)**.** A disaggregation $D = (m_D, a_D, \tilde{b}_D)$ is given by a number of tasks $m_D \in \mathbb{N}^*$, a map $a_D : [\![1, m_D]\!] \to [\![1, m]\!]$ associating each new task of $[\![1, m_D]\!]$ with an original task of $[\![1, m]\!]$ and a new right-hand side $\tilde{b}_D : [\![1, m_D]\!] \to \mathbb{N}^*$. In addition, the disaggregation D must preserve the number of task coverages. It must therefore verify: $\forall i \in [\![1, m]\!], \sum_{k \in \tilde{W}_D(i)} \tilde{b}_D(k) = b_i$, where $\tilde{W}_D(i) = a_D^{-1}[\{i\}]$ is the inverse image of $\{i\}$ by a_D function, the set of new tasks associated with old task i.

In Example 1, the disaggregation suggested above is described by $m_D = 4$, the map a_D defined by $a_D(1) = a_D(2) = 1$ and $a_D(3) = 2, a_D(4) = 3$, and right-hand side $\tilde{b}_D$ with $\tilde{b}_D(1) = \tilde{b}_D(2) = 1$ and $\tilde{b}_D(3) = \tilde{b}_D(4) = 2$. The coverage number between the initial task and its associated new tasks remains the same: the first task, initially to be carried out by two people, is replaced by two tasks, each to be carried out by one person. Note that we would have $\tilde{W}_D(1) = \{1, 2\}$, $\tilde{W}_D(2) = \{3\}$, and $\tilde{W}_D(3) = \{4\}$.

4.2 Equivalent disaggregated problem

From a valid task disaggregation D, it is possible to construct an equivalent problem: $GSPP_D^{dis}$. This problem is equivalent to GSPP in the sense that there is an explicit cost-preserving surjection between

its integer solutions and those of the original problem. The two problems then have the same objective, and it is easy to obtain an integer solution of $GSPP$ from an integer solution of $GSPP_D^{dis}$.

In Example 1, after disaggregation, the number of tasks becomes 4 instead of 3. For a column A_j of the original problem, there are two possibilities: either A_j covers the disaggregated task or it does not. For example, $A_1 = \begin{pmatrix}1 & 0 & 1\end{pmatrix}^T$ covers the first task, which has two equivalents in the disaggregated problem. To construct its associates in the disaggregated problem, we can therefore duplicate the column into two new columns: $\begin{pmatrix}1 & 0 & 0 & 1\end{pmatrix}^T$ and $\begin{pmatrix}0 & 1 & 0 & 1\end{pmatrix}^T$. Each duplicate of the column covers a different copy of the disaggregated task. Conversely, a column that doesn't cover a disaggregated task, such as $A_2 = \begin{pmatrix}0 & 1 & 0\end{pmatrix}^T$, can be replaced by a single column that doesn't cover any of the disaggregated tasks: $\begin{pmatrix}0 & 0 & 1 & 0\end{pmatrix}^T$. Applying this same process to all columns of A, the new constraint matrix $\tilde{A}$ and right-hand side $\tilde{b}$ are as follows:

$$\tilde{A} = \begin{pmatrix}1 & 0 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 1 & 1 & 1\\ 1 & 1 & 0 & 1 & 0 & 0\end{pmatrix}, \tilde{b} = \begin{pmatrix}1\\1\\2\\2\end{pmatrix} \tag{9}$$

In this new definition, columns 1 and 2 of $\tilde{A}$ are associated with the original column 1 of A, column 3 of $\tilde{A}$ with the original column 2 of A, column 4 of $\tilde{A}$ with the original column 3 of A and columns 5 and 6 of $\tilde{A}$ with the original column 4 of A. Each column i of $\tilde{A}$ is associated with a decision variable $\tilde{x}_i$. To make the decomposed problem equivalent to the original one, it is necessary to further impose that for each original column, at most one of its copies is selected. In Example 1, this can be done by adding the constraints $\tilde{x}_1 + \tilde{x}_2 \leq 1$, $\tilde{x}_3 \leq 1$, $\tilde{x}_4 \leq 1$ and $\tilde{x}_5 + \tilde{x}_6 \leq 1$. We then obtain the following problem:

$$\begin{aligned}
&\min\ c_1\tilde{x}_1 + c_1\tilde{x}_2 + c_2\tilde{x}_3 + c_3\tilde{x}_4 + c_4\tilde{x}_5 + c_4\tilde{x}_6\\
&\ s.t.\\
&\quad \tilde{A}\tilde{x} = \tilde{b}\\
&\quad \tilde{x}_1 + \tilde{x}_2 \leq 1\\
&\quad \tilde{x}_3 \leq 1\\
&\quad \tilde{x}_4 \leq 1\\
&\quad \tilde{x}_5 + \tilde{x}_6 \leq 1\\
&\quad \tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6 \in \{0,1\}
\end{aligned}$$

All new columns remain binary and have the same number of non-zero elements in matrix $\tilde{A}$ as their original column had in matrix A. In other words, for a column of the disaggregated problem, the sum of the contributions of the new tasks 1 and 2 is equal to the contribution of the original task 1. To translate vectors (matrix columns, solutions, etc.) from the disaggregated problem to the original one, we define a *sum map*. For a valid disaggregation $D = (m_D, a_D, \tilde{b}_D)$, this sum map is denoted by $S_D : \{0,1\}^{m_D} \to \{0,1\}^m$. Let $v \in \mathbb{R}^{m_D}$. Then, $S_D(v)_i = \sum_{k \in \tilde{W}_D(i)} v_k$ (all elements corresponding to the same initial task are summed). In particular, if v is a column of $\tilde{A}$, then $S_D(v)$ corresponds to the equivalent column in A. In Example 1, columns $\tilde{A}_1 = \begin{pmatrix}1\\0\\0\\1\end{pmatrix}$ and $\tilde{A}_2 = \begin{pmatrix}0\\1\\0\\1\end{pmatrix}$ are such that $S_D(\tilde{A}_1) = S_D(\tilde{A}_2) = A_1 = \begin{pmatrix}1\\0\\1\end{pmatrix}$ (summation of the first two components, keeping the last

two invariant). $S_D^{-1}[\{A_j\}]$ corresponds to the set of all binary columns of $\{0,1\}^{m_D}$ which have the column A_j as an associate.

We now formally define the equivalent disaggregated problem in the general case. Let $j \in [\![1,n]\!]$. The columns of $S_D^{-1}[\{A_j\}]$ can be ordered in any order to obtain M_D^j, an $n_D^j = Card(S_D^{-1}[\{A_j\}])$-tuple. Let $(M_D^j)_k$ be the k-th element of this n_D^j-tuple, for $k \in [\![1, n_D^j]\!]$ (corresponding to the k-th copy of A_j in the disaggregated problem). Similarly, let $\tilde{x}_k^j$ be a binary decision variable associated with the $k \in [\![1, n_D^j]\!]$-th copy of the original decision variable x_j. The equivalent disaggregated problem $GSPP_D^{dis}$ is as follows:

$$\min \sum_{j=1}^{n} c_j \sum_{k=1}^{n_D^j} \tilde{x}_k^j \tag{10}$$

$$s.t.$$

$$\sum_{j=1}^{n} \sum_{k=1}^{n_D^j} \tilde{x}_k^j (M_D^j)_k = \tilde{b}_D \tag{11}$$

$$\sum_{k=1}^{n_D^j} \tilde{x}_k^j \leq 1 \; \forall j \in [\![1,n]\!] \tag{12}$$

$$\tilde{x}_k^j \in \mathbb{N} \; \forall j \in [\![1,n]\!], \forall k \in [\![1, n_D^j]\!] \tag{13}$$

The following two propositions 1, 2 show that integer solutions of $GSPP$ and $GSPP_D^{dis}$ are equivalent.

Proposition 1. Let $\tilde{x} \in Dom(GSPP_D^{dis})$ be a feasible integer solution of $GSPP_D^{dis}$. Then the vector $\bar{x} \in \mathbb{R}^n$ defined by $\bar{x}_j = \sum_{k=1}^{n_D^j} \tilde{x}_k^j$ is also a feasible integer solution of $GSPP$ of same cost.

Proof. We have $\sum_{j=1}^{n} \sum_{k=1}^{n_D^j} \tilde{x}_k^j (M_D^j)_k = \tilde{b}_D$. We note that $\forall j \in [\![1,n]\!]$, $\forall k \in [\![1, n_D^j]\!], S_D((M_D^j)_k) = A_j$. Applying the linear sum map S_D (by extending its domain of definition to $\mathbb{R}^n$) on either side of the equality, we obtain: $\sum_{j=1}^{n} A_j (\sum_{k=1}^{n_D^j} \tilde{x}_k^j) = \sum_{j=1}^{n} A_j \bar{x}_j = b$. Moreover, constraints (12) indicate that $\bar{x}_j \leq 1$, $\forall j \in [\![1,n]\!]$. $\bar{x}$ is therefore feasible for $GSPP$ and has the same cost as $\tilde{x}$. □

To show that for $\bar{x}$, a feasible solution of $GSPP$, there exists $\tilde{x} \in Dom(GSPP_D^{dis})$ such that $\forall j \in [\![1,n]\!]$, $\sum_{k=1}^{n_D^j} \tilde{x}_k^j = \bar{x}_j$, we first prove Lemma 1 in Appendix B. The statement and proof of the lemma are in Appendix B. Thanks to this lemma, we can now prove the following proposition:

Proposition 2. Let $\bar{x}$ be a feasible integer solution of $GSPP$. Then there exists $\tilde{x} \in Dom(GSPP_D^{dis})$ of equal cost verifying $\forall j \in [\![1,n]\!] \sum_{k=1}^{n_D^j} \tilde{x}_k^j = \bar{x}_j$.

Proof. The construction of $\tilde{x}$ is done step by step. We denote $\{j \in [\![1,n]\!] | \bar{x}_j = 1\} = \{j_1, .., j_p\}$. We construct $\tilde{x}$ by associating each index j_k for $k \in [\![1,p]\!]$ to one of its copies in the disaggregated problem, $f(j_k) \in [\![1, n_D^{j_k}]\!]$. For the first index j_1, we associate $f(j_1) \in [\![1, n_D^{j_1}]\!]$ with any column index associated with column j_1. We can verify that $(M_D^{j_1})_{f(j_1)} \leq \tilde{b}_D$ (since the column $(M_D^{j_1})_{f(j_1)}$ is binary and $\tilde{b}_D$ has strictly positive integer coefficients). We set $\tilde{x}_{f(j_1)}^{j_1} = 1$. Suppose that for $k \in [\![1, p-1]\!]$, $f(j_1)$

through $f(j_k)$ are constructed. $v = \sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)}$ represents the total coverage number of the columns of $GSPP_D^{dis}$ already selected. We also assume that $v \leq \tilde{b}_D$ (this is initially true when $k = 1$). We see, by Lemma 1 in Appendix B, that for every original task i covered by the column $A_{j_{k+1}}$, there exists a task $i' \in \tilde{W}_D(i)$ such that $v_{i'} < (\tilde{b}_D)_{i'}$. We can therefore construct $f(j_{k+1})$ by choosing the index of an element of $S_D^{-1}[\{A_{j_{k+1}}\}]$ which covers exactly one task $i' \in \tilde{W}_D(i)$ such that $v_{i'} < (\tilde{b}_D)_{i'}$ for each original task i covered by the column $A_{j_{k+1}}$. We then set $\tilde{x}^{j_{k+1}}_{f(j_{k+1})} = 1$ and indeed have $\sum_{l=1}^{k+1} (M_D^{j_l})_{f(j_l)} \leq \tilde{b}_D$ (since the column $(M_D^{j_{k+1}})_{f(j_{k+1})}$ is binary).

Once all the $f(j_k)$ have been constructed for each $k \in [\![1, p]\!]$, all other components of $\tilde{x}$ are taken to be zero. The coverage constraints (11) are all satisfied by $\tilde{x}$ because an original task $i \in [\![1, m]\!]$ is covered exactly b_i times by the columns $\{A_{j_1}, ..., A_{j_p}\}$; if one of the coverage constraints (11) were not satisfied, there would exist a task $i' \in [\![1, m_D]\!]$ such that $(\sum_{l=1}^{p} (M_D^{j_l})_{f(j_l)})_{i'} < (\tilde{b}_D)_{i'}$. We would therefore have for the original task $i = a_D(i')$ (associated with the task i'), $(\sum_{l=1}^{p} A_{j_l})_i < b_i$ whereas $(\sum_{l=1}^{p} A_{j_l})_i = b_i$. We can therefore conclude that all coverage constraints (11) are satisfied by $\tilde{x}$. Furthermore, since at most one column is selected in $[\![1, n_D^j]\!]$ for each $j \in [\![1, n]\!]$, the bound constraints (12) are also verified. $\tilde{x}$ is therefore feasible for $GSPP_D^{dis}$. We then verify that we have $\forall j \in [\![1, n]\!] \sum_{k=1}^{n_D^j} \tilde{x}_k^j = \overline{x}_j$ and that the costs of the solutions are the same. □

Together, Propositions 1 and 2 show that there is indeed a linear surjection preserving the cost between the domains of $GSPP_D^{dis}$ and $GSPP$. We denote the existence of this surjection by the following notation:

$$Dom(GSPP_D^{dis}) \twoheadrightarrow Dom(GSPP)$$

Furthermore, note that the disaggregated problems also have the structure of a $GSPP$ (by adding binary slack variables to the constraints of type "≤ 1"). The stability of this structure after disaggregation implies that resolution techniques similar to those used on $GSPP$ can also be used on $GSPP_D^{dis}$. This common structure includes cover constraints, bounds constraints, and a constraint matrix with binary coefficients.

The number of bounds constraints (12) remains equal to n and each bounds constraint $j \in [\![1, n]\!]$ of $GSPP_D^{dis}$ corresponds to a subset of variables: the variables associated with column j. These subsets partition the variables of $GSPP_D^{dis}$. The number of cover constraints increases to $\sum_{i=1}^{m} |\tilde{W}_D(i)| = m_D > m$ and the number of columns increases accordingly. In Example 1, each column covering task 1 is duplicated. On the other hand, the proximity of GSPP to SPP increases with the size of the disaggregation characterized by m_D. The results of a resolution strategy in this disaggregated problem could therefore be similar to those in an SPP.

This proximity can be measured by the sum of the distances of each right-hand side to 1: $d_{SPP}(D) = \sum_{i=1}^{m_D} (\tilde{b}_D(i) - 1)$. An invariant of task disaggregation is the sum of the right-hand sides, which remains equal to the total number of covers $\sum_{i=1}^{m} b_i$. This distance, $d_{SPP}(D)$, is then written as $(\sum_{i=1}^{m} b_i) - m_D = d_{SPP}(D_I) - (m_D - m)$, where D_I corresponds to the identity disaggregation that gives back the $GSPP$ problem. The greater the number of constraints m_D, the smaller this distance. $m_D - m$ is therefore a good indication of the contribution of disaggregation D compared to the original $GSPP$ problem. The maximum disaggregation (all tasks are fully disaggregated, and the final right-hand sides are therefore all 1) is denoted D_{max}. Note that $d_{SPP}(D_{max}) = 0$.

When tasks are fully disaggregated, the $GSPP$ is therefore reduced to an equivalent problem that is an SPP. D_{max} corresponds to this disaggregation. The size of $GSPP_{D_{max}}^{dis}$ is bounded by a

function of the size of $GSPP$. Let $b_{max} = \max_{i \in [\![1,m]\!]} \{b_i\}$ and K be the maximum number of tasks covered by a column. For $GSPP^{dis}_{D_{max}}$, a task with right-hand side b_{max} now has b_{max} equivalent tasks, and there are therefore at most $(b_{max})^K$ columns associated with the same initial column of $GSPP$ in the maximal disaggregated problem. $GSPP^{dis}_{D_{max}}$ therefore has at most nb^K_{max} columns and mb_{max} tasks (not counting the bounds constraints). As a function of b_{max} and K, the reduction from GSPP to SPP is therefore polynomial. For the problems treated in this article, we can estimate the factor $(b_{max})^K$ at $4^7 = 16,384$ (4 corresponding to b_{max} and 7 to the average number of tasks covered by a column). This factor is, for example, smaller than the number of columns n in our problems. However, in our practical method, only certain tasks are actually disaggregated, and a filter is applied to the columns of the disaggregated problem. Consequently, in our results in Section 6.5, we observe that the number of columns in the disaggregated problem remains of the order of n.

4.3 Symmetries of the disaggregated problem

In this subsection, we present a way to further reduce the number of columns of the disaggregated problem. One way to reduce the size of an object is to eliminate its symmetries: knowing the symmetries and the reduced object (so that it no longer has the symmetries) allows us to reconstruct the original object. For a certain disaggregation D and an initial task $i \in [\![1, m]\!]$, it is easy to observe that all the tasks associated with i in the disaggregated problem ($\tilde{W}_D(i)$) are equivalent since for an initial column A_j, all binary columns of $S_D^{-1}[\{A_j\}]$ are associated with it. The permutations of tasks of $\tilde{W}_D(i)$ then constitute a set of symmetries of the constraint matrix (cover constraints (11) without bound constraints (12)). After permuting tasks from $\tilde{W}_D(i)$ (equivalent tasks associated with the same original task i) in the constraint matrix, the constraint matrix remains unchanged: it is possible to re-index the variables (or reorder the columns) in order to obtain the same constraint matrix (constraint matrix before permutation).

It is possible to construct an equivalent problem to $GSPP^{dis}_D$ that no longer has these symmetries. This problem is constructed by reducing $GSPP^{dis}_D$ with respect to a single feasible solution (initial or current solution, for example). To do this, we consider a feasible integer solution $\bar{x}$ of $GSPP$ that covers the initial columns indexed by $\mathcal{P} = \{j_1, ..., j_p\}$. Each $j \in [\![1, n]\!]$ has several associated column indices in the disaggregated problem $GSPP^{dis}_D$. We denote these indices by F_j. Using the same constructive procedure as in Proposition 2, we can construct an equivalent point $\tilde{x}$ in $GSPP^{dis}_D$ such that only one element of F_{j_k}, denoted $f(j_k)$, is selected for each $k \in [\![1, p]\!]$.

The reduced disaggregated problem is defined as follows: for columns $j_k \in \mathcal{P}$, we keep only column $f(j_k)$ and remove all other columns of F_{j_k}. For $j \notin \mathcal{P}$, all columns of $S_D^{-1}[\{A_j\}]$ are kept.

To illustrate how symmetry reduction works, we apply it to the following example where all tasks are disaggregated.

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

Suppose the current solution is $\bar{x} = (\mathcal{P}, \mathcal{Z})$ with $\mathcal{P} = \{1, 2, 4\}$ and $\mathcal{Z} = \{3\}$. Following disaggregation, we have:

$$\tilde{A} = \left(\begin{array}{cccc|cc|cccc|cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right)$$

Where the dashed lines indicate the relations between the original tasks/columns and the new ones. The highlighted columns represent the columns of $\tilde{\mathcal{P}}$ (active columns of the solution $\tilde{x}$ in the disaggregated problem associated with the solution $\overline{x}$) selected during the symmetry reduction procedure. After symmetry reduction, in the disaggregated problem reduced by $\overline{x}$, $\tilde{A}$ is reduced to:

$$\tilde{A} = \left(\begin{array}{c:cc:cc:cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hdashline 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hdashline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

There are no longer any symmetries corresponding to permutations of new equivalent tasks (new tasks associated with the same original task). For example, it is no longer possible to permute task 1 and task 2 without changing the constraint matrix (up to column permutation), whereas this was possible for the matrix $\tilde{A}$ (up to column permutation).

This new disaggregated problem (GSPP_D^{dis} reduced by x) is still equivalent to the initial problem but no longer has the symmetries corresponding to the permutations of equivalent tasks, as shown in the following proposition.

Proposition 3. *$GSPP_D^{dis}$ reduced by a feasible integer solution $\bar{x}$ of $GSPP$ is equivalent to $GSPP_D^{dis}$ but no longer has the symmetries corresponding to the permutations of tasks associated with the same original task.*

Proof. Each feasible solution of $GSPP_D^{dis}$ reduced by $\bar{x}$ still has an equivalent of the same cost in the original problem. It is sufficient to verify that there is still a surjection between $Dom(GSPP_D^{dis})$ and $Dom(GSPP)$. If we consider a solution x' of $GSPP$, we can, as in Proposition 2, construct an equivalent point $\tilde{x}'$ in $GSPP_D^{dis}$. It is sufficient to start the order of traversal of the columns covered by x' with the columns in $\{j_1, ..., j_p\}$ and to favor the unique column $f(j_k)$ for such columns. The two problems are therefore equivalent. Furthermore, if we choose $i \in [\![1, m]\!]$, it is no longer possible to permute two tasks from $\tilde{W}_D(i)$ without changing the problem. The solution $\overline{x}$ actually covers all tasks. If two tasks from $\tilde{W}_D(i)$ are permuted at the level of each column, for an active column j_k of the solution $\overline{x}$ that covers task i, the column associated with $f(j_k)$ is modified. However, if we suppose that the columns A_j for $j \in [\![1, n]\!]$ are all distinct, this modified column no longer appears in the columns of the problem, since in $GSPP_D^{dis}$ reduced by $\overline{x}$, only one column has been retained from among the columns associated with column j_k (column $f(j_k)$). If the columns of the original matrix A are all distinct, it is then no longer possible after task permutation to reorder the columns (or reindex the variables) in order to obtain the same constraint matrix. Then, $GSPP_D^{dis}$ no longer has the symmetries corresponding to the permutations of tasks associated with the same original task (assuming that the columns of matrix A are all distinct). □

5 Disaggregation algorithm

At the CP level of GISUD, a normalized direction of minimal reduced cost is selected by solving a linear program. This direction can be integer or fractional. By using task disaggregation, the success rate of this complementary problem can be increased through successive resolution of complementary problems, each having lower right-hand sides than the previous one. Adding this disaggregation component to the standard GISUD leads to a version of the GISUD algorithm, which we call $GISUD^{dis}$. This section presents the disaggregation component of $GISUD^{dis}$. An overview of this component is first presented in Section 5.1. Sections 5.2 and 5.3 detail its different parts. Section 5.4 provides a proof of convergence of the algorithm.

5.1 Overview of the algorithm

The following algorithm replaces the CP of GISUD. A flowchart of the algorithm is shown in Figure 3.

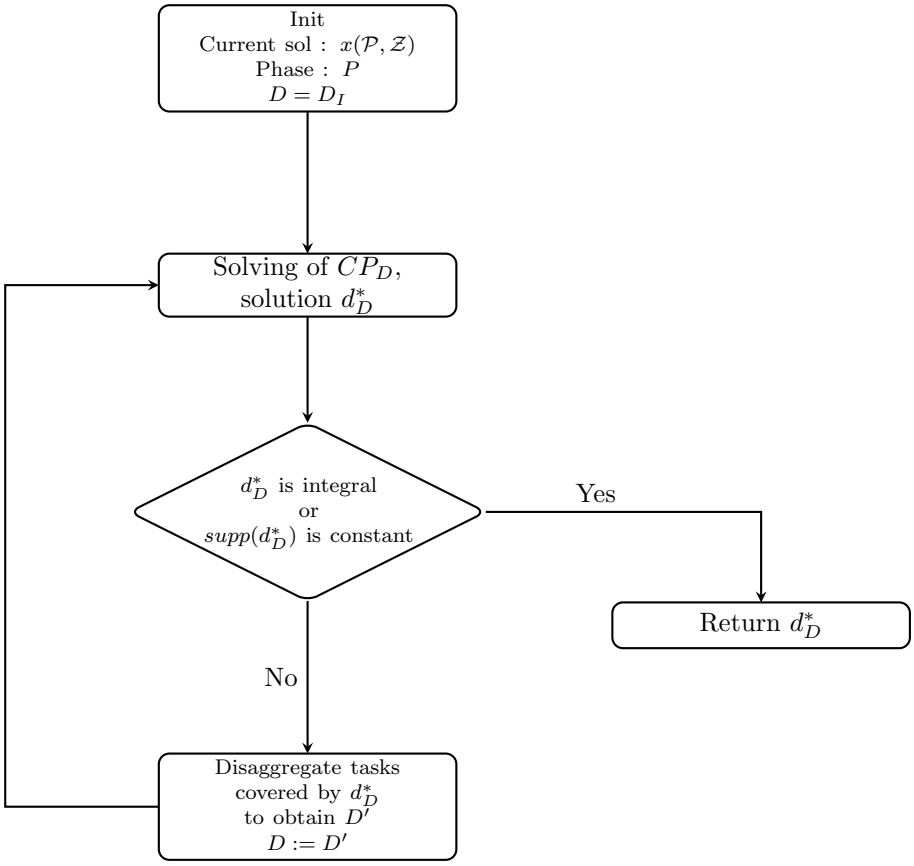

Figure 3: Flowchart of the CP in GISUD^{dis}

It takes as input a current solution x represented by $(\mathcal{P}, \mathcal{Z})$, as well as a phase P that limits the incompatibility degree of the columns considered. The purpose of this algorithm is to return, if possible, an improving integer direction. The observation underlying the algorithm is that maximum disaggregation D_{max} leads to an equivalent problem where all right-hand sides are 1 (as in the SPP). The success rate of the CP at a point equivalent to x in $GSPP^{dis}_{D_{max}}$, which we denote $\tilde{x}^{max}$, would then, in phase P, more closely resemble that of a complementary problem of an SPP in phase P (which has a good success rate). Our algorithm then computes a sequence of directions that converges towards the solution direction of this maximum complementary problem CP in phase P without necessarily needing to disaggregate all the tasks. To do this, the CP is first solved in phase P in order to find a normalized direction d^* of minimum reduced cost starting from x. If this direction is integer, it is returned. Otherwise, the tasks covered by at least one column of $supp(d^*)$ are stored in the set $W = \{i \in [\![1, m]\!] | \exists j \in supp(d^*), A_{ij} = 1\}$. These tasks are completely disaggregated, replacing each task $i \in W$ that has right-hand side b_i with b_i tasks with right-hand side 1. The other tasks are not disaggregated. This gives a disaggregation D that leads to an equivalent problem as seen in subsection 4.2. A point $\tilde{x}$ equivalent to x is chosen from $GSPP^{dis}_D$. A complementary disaggregated problem (CP_D) similar to the classic CP is then solved at point $\tilde{x}$. Only the columns of $GSPP^{dis}_D$ that have a phase smaller than P are considered in the CP_D. If the solution direction d_D^* of CP_D leads to an integer solution, this direction is returned. Otherwise, the process is repeated by adding to W tasks covered by the columns of $supp(d_D^*)$. A new disaggregation D' is defined and $CP_{D'}$ is solved. The procedure stops once d_D^* is integer or if $supp(d_D^*)$ is the same for two consecutive iterations. This

termination criterion will be justified in Section 5.4. When the procedure terminates, an equivalent direction in $GSPP$ is returned. If this direction is integer, the solution is updated and a new iteration begins; otherwise, the ZOOM procedure takes over.

5.2 Disaggregated complementary problem

Let $\bar{x} = (\mathcal{P}, \mathcal{Z})$ be the current solution, with $\tilde{x}$ an equivalent solution in $GSPP_D^{dis}$. The modified CP for $GSPP_D^{dis}$, CP_D, is a linear program that aims to obtain an integer improvement direction from $\tilde{x}$ in $GSPP_D^{dis}$. CP_D thus has the same structure as CP. We present here the mathematical model for CP_D, which uses the notation for the disaggregation outlined in Section 4.

In $GSPP_D^{dis}$, each column $j \in \mathcal{Z}$ is represented by n_D^j columns $(M_D^j)_1, ..., (M_D^j)_{n_D^j}$. We choose to remove the symmetries of $GSPP$ by reducing it with respect to the current solution $\bar{x}$. Recall that once symmetries are removed using the current solution $\bar{x}$, each column $j \in \mathcal{P}$ is represented by a single column, denoted $f(j)$ in $GSPP_D^{dis}$.

The direction from $\tilde{x}$ to another point of the polytope $\tilde{z} \in Dom(GSPP_D^{dis})$ is denoted $d = \tilde{z} - \tilde{x}$. The components of $supp(d)$ on the positive columns of $\tilde{x}$ are negative and they are positive on the other columns. Similarly to the CP of GISUD, CP_D has as its solution the normalized direction of minimal reduced cost. CP_D is formulated as follows:

$$CP_D(\mathcal{P}, \mathcal{Z}) = \min \sum_{j \in \mathcal{P}} c_j d^j_{f(j)} + \sum_{j \in \mathcal{Z}} c_j \sum_{k=1}^{n_D^j} d^j_k \tag{14}$$

$$s.t.$$

$$\sum_{j \in \mathcal{P}} d^j_{f(j)} (M_D^j)_{f(j)} + \sum_{j \in \mathcal{Z}} \sum_{k=1}^{n_D^j} d^j_k (M_D^j)_k = 0 \tag{15}$$

$$- \sum_{j \in \mathcal{P}} w^j_{f(j)} d^j_{f(j)} + \sum_{j \in \mathcal{Z}} \sum_{k=1}^{n_D^j} w^j_k d^j_k = 1 \tag{16}$$

$$d^j_{f(j)} \leq 0 \; \forall j \in \mathcal{P} \tag{17}$$

$$d^j_k \geq 0 \; \forall j \in \mathcal{Z} \; \forall k \in [\![1, n_D^j]\!] \tag{18}$$

Constraints (15), (17), and (18) are the feasibility constraints of a direction starting from $\bar{x}$. They correspond respectively to constraints (5), (7), and (8) of the classical complementary problem $CP(\mathcal{P}, \mathcal{Z})$. All weights w^j_k and $w^j_{f(j)}$ involved in constraint (16) are strictly positive. Constraint (16) thus bounds the cone of feasible directions in the same way as constraint (6) in the classical complementary problem. CP_D is similar to CP but operates on the disaggregated problem reduced by $\bar{x}$. The following proposition indicates that each integer direction of the original CP admits an equivalent integer direction in CP_D.

Proposition 4. Let d be an integer direction of $Dom(CP)$ with negative reduced cost. There exists a direction $\tilde{d}$ of $Dom(CP_D)$ equivalent to d with negative reduced cost.

Proof. d connects integer point $\bar{x}$ to another integer point $\bar{z}$. $\tilde{x}$ is equivalent to $\bar{x}$ in $GSPP_D^{dis}$, and let $\tilde{z}$ be an equivalent of $\bar{z}$ in $GSPP_D^{dis}$. According to the constructive algorithm outlined in Proposition 2, we can choose $\tilde{z}$ so that it doesn't involve any column associated with $j \in \mathcal{P}$ other than $f(j)$. The direction $\tilde{z} - \tilde{x}$ is therefore in the direction cone and provides a direction of $Dom(CP_D)$ with strictly negative reduced cost when normalized with weights w. □

Finally, note that CP_D is compatible with the multi-phase procedure outlined in Section 3. Indeed, variable $d^j_k, j \in \mathcal{Z}$ is only included if the incompatibility degree of its corresponding column is lower

than or equal to P, the current phase. The next section presents an efficient procedure for generating such columns.

5.3 Generation of columns for the complementary problem

An initial column of $\mathcal{Z}$ of GSPP has several associated columns in $GSPP_D^{dis}$. In CP_D, only those columns whose phase is smaller than P appear. In order to generate these columns, the first instinct is to generate all the columns associated with each column $j \in \mathcal{Z}$, compute their incompatibility degree, and keep only those with a phase smaller than P. This calculation can be time-consuming, given that many columns can be associated with the same column. The second possibility is to directly generate columns with a phase smaller than P. This procedure will be illustrated with the example in Section 3.1: The tasks are in order 1, 2, 3.

$$A = \left(\begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}\right) \text{ and } b = \begin{pmatrix}1\\2\\2\end{pmatrix}$$

We assume that $\mathcal{P}$ corresponds to the columns $\{1, 2, 3\}$ of matrix A and that $k = 4$. Consider the disaggregation $D = (m_D, a_D, r_D)$ with $m_D = 5$, $a_D(1) = 1, a_D(2) = a_D(3) = 2, a_D(4) = a_D(5) = 3$ and $r_D(k) = 1, \forall k \in [\![1, 5]\!]$ (all tasks are disaggregated). We want to generate the columns associated with column A_k with an incompatibility degree less than P in the disaggregated problem. To do this, we use the same graph as the one used to calculate the degree of incompatibility of column A_k in the GISUD algorithm (Figure 1). Recall that in that graph, node (i, j) was associated with task i and column j of $\mathcal{P}$.

Although the graph structure remains identical, the interpretation of its node is different here. Node (i, j) is now associated with copy j of task $i \in W_k$ in the dissaggregated problem. The number of nodes in the graph is thus equal to m_D (5 in the example).Each path in the graph is then associated with a column in the disaggregated problem, where the nodes visited indicate which copies of the original tasks are selected. The length of the path corresponds to an upper bound on the incomatibility degree of the associated column in $GSPP_D$. Details of this association are provided in the Appendix A. It is therefore possible to enumerate all paths shorter than P directly in this graph in order to generate columns with an incompatibility degree less than P.

In practice, it has been observed that there are few paths shorter than P. Enumerating these paths is therefore quick (unlike enumerating all possible columns in the disaggregated problem).

5.4 Choice of weights and convergence of the procedure

The weights of normalization constraint (16) are the parameters of the CP_D. There is one weight for each column of CP_D. They must therefore be chosen for the new columns $[\![1, n_D^j]\!]$ associated with each column $j \in [\![1, n]\!]$.

We show that the following criterion are sufficient for the convergence of the method:

- $w_k^j = w_j$ if $n_D^j = 1$ (if the column is not duplicated, its weight is unchanged)
- $w_k^j \geq w_j \ \forall k \in [\![1, n_D^j]\!]$ otherwise.

Similarly, when disaggregating tasks from D to obtain D', the columns of $CP_{D'}$ can be seen as associated with columns of CP_D. The same criterion also apply. This choice of weights allows the costs of successive CP_D problems to be increasing, which is necessary for the following demonstrations. The proposition below indicates that if D' is a disaggregation obtained by further disaggregating tasks from D, then the objective of $CP_{D'}$ is greater than or equal to the objective of CP_D.

Proposition 5. Let D be a disaggregation. Let D' be a disaggregation obtained by further disaggregating tasks from D. Then each feasible solution d' of $CP_{D'}$ with negative cost admits an equivalent representative in CP_D with a smaller or equal cost.

Proof. Consider a feasible solution d' of $CP_{D'}$. The key point is that each column of $CP_{D'}$ can be seen as associated with a column of CP_D because D is a smaller disaggregation than D'. By summing the components of d' on the columns associated with the same column of CP_D, we obtain a new vector d of the same dimension as the solution space of CP_D. This vector d remains feasible (without the normalization constraint) because the function S_D is linear (similar argument to that of Proposition 1). It remains to normalize the vector d. Its norm is smaller or equal to 1 because the weights are increasing and the norm of d' with the weights of $CP_{D'}$ is equal to 1. Since d' has a negative cost, we deduce that the cost of the normalized d in CP_D is smaller or equal to that of d' in $CP_{D'}$. □

We can deduce the following proof of convergence: When the support of the CP_D solution remains constant between two successive disaggregations, the current solution corresponds to the optimal solution of $CP_{D_{max}}$.

Proposition 6. Let d^*_D be the optimal solution of CP_D. If by disaggregating the tasks covered by d^*_D to obtain disaggregation D', the optimal solution of $CP_{D'}$ remains unchanged, then d^*_D is optimal for $CP_{D_{max}}$.

Proof. d^*_D is feasible for $CP_{D_{max}}$ because it is feasible for $CP_{D'}$ with disaggregation D' that has all tasks covered by d^*_D disaggregated (columns of $supp(d^*_D)$ are not disaggregated between disaggregation D' and disaggregation D_{max}). The optimal objective of $CP_{D_{max}}$ is greater than that of $CP_{D'}$. However, since in disaggregation D', all tasks covered by d^*_D are disaggregated, there is no further disaggregation of these tasks between D' and D_{max}. The weights remain the same on the columns covered by d^*_D between $CP_{D'}$ and $CP_{D_{max}}$. The cost of d^*_D therefore remains the same between $CP_{D'}$ and $CP_{D_{max}}$. Being lower than the optimal objective of $CP_{D_{max}}$, we deduce that d^*_D is optimal for $CP_{D_{max}}$. □

6 Results

This section presents the experiments of the GISUD method with task disaggregation for aircrew pairing problems. Subsection 6.1 presents the experimental protocol used and subsection 6.2 the characteristics of the different tested instances. Subsections 6.3 and 6.4 then present the experimental results with their discussion. Performing disaggregation on a problem increases its size (number of variables and number of constraints). In order for the disaggregation process to be as fast as possible (even though its resolution consists solely of solving a linear program), the disaggregated problem size must remain reasonable. We therefore verify, in subsection 6.5, that the size of the disaggregated problems (computed by the algorithm $GISUD^{dis}$) remains reasonable.

6.1 Experimental protocol

GISUD^{dis} was run on 8 different instances. The CPLEX MIP solver was also run on the same instances. All experiments are conducted on the test instances described in Section 6.2 using a Linux computer with an Intel i7-8700 CPU 3.20GHz. The method was implemented in C++. For both solvers, 8 threads were used.

Although GISUD^{dis} is an exact method, solving large-scale instances of the GSPP would take too much time for either method. For this reason, we stop resolution when a feasible solution with an optimality gap of less than 0.5% is found. For CPLEX MIP, the optimality gap is naturally computed during resolution. For GISUD^{dis}, we start the algorithm by solving the linear relaxation of the problem, which provides a lower bound, that is used to approximate the optimality gap. For the instances solved, this bound is empirically of good quality. Note that this procedure overestimates the optimality gap, so GISUD^{dis} is slightly disadvantaged in that regard.

GISUD^{dis} requires an initial solution. We experiment with two ways of generating such initial solutions. We first test with solution created from scratch using a procedure described in Section 6.2

(fast procedure). Another real-life application of GISUD^{dis} would be in a reoptimization context, for instance when reoptimizing an aircrew schedule following an airport closure. In that case, a solution is generally available but no longer optimal. To simulate this use case, we also create perturbed solution from a near optimal solution. The perturbation procedure is the same as in Barry et al. (2024) but is also summarized in Section 6.2. In all cases, the initial solutions are also provided to the CPLEX MIP solver.

Regarding the weights of the normalization constraint, for a column j, we chose $w_j = \hat{d}_{j,\mathcal{P}} + 1$ (with $\hat{d}_{j,\mathcal{P}}$ corresponding to the incompatibility degree of column j with respect to the current solution $\mathcal{P}$). This ensures that w_j remains strictly positive. This also ensures that upon disaggregating, the weight of the disaggregated columns satisfy the criterion enunciated in Section 5.4 (that the weight of the copies are greater or equal to that of their original). Indeed, it is easy to show that the incompatibility degree of a disaggregated column is always greater or equal than that of the original.

For practical reasons, at each CP failure, the CP problem is disaggregated once into CP_D. If CP_D returns a failure, the disaggregation strategy returns a failure instead of iterating the disaggregation. Preliminary results showed that doing this is faster than iteratively disaggregating. Finally, we only attempt the disaggregation procedure if the CP has fewer than a fixed number of variables (so disaggregation mainly occurs in small phases). This is to keep the disaggregated problem at a reasonable size.

6.2 Instances

The tests were conducted on GSPPs extracted from Barry et al. (2024), and presented in Table 1. These are SPP instances modified to include right-hand side values other than 1. Two distinct right-hand side distributions are considered. In the first (RHS1), values are uniformly distributed between 1 and 3. In the second (RHS2), they are uniformly distributed between 1 and 4. For each instance, Table 1 lists the RHS type, the number of variables, the number of tasks, the average number of task covered by each variable (the number of non-zero values), and the exact right-hand side value distribution. More details on the generation of the instances are presented in Barry et al. (2024).

Table 1: Table of instances

Instance	RHS type	N cols	N tasks	N zeros/col	Right-hand sides			
					1	2	3	4
320_1	RHS1	263 289	1990	4.61	675	657	658	0
320_2	RHS1	228 859	2214	4.79	753	729	732	0
320_3	RHS2	387 077	1990	4.55	494	499	500	497
320_4	RHS2	372 575	2214	4.85	553	553	553	555
757_1	RHS1	234 770	2020	4.77	688	666	666	0
757_2	RHS1	274 856	2021	4.84	689	664	668	0
757_3	RHS2	362 487	2020	4.91	504	505	504	507
757_4	RHS2	321 975	2017	4.95	504	503	504	506

We use two methods to generate initial solutions. The first is to create one from scratch using the following procedure. First, the linear relaxation of the problem is solved using a standard linear programming algorithm (CPLEX). The variables are then ordered by decreasing order of their value in the LP solution. To build a solution, variables are fixed to 1 according to that order, ignoring the variables that become infeasible in the process. Finally, slack variables with a very high cost (1,000,000) are added to complete the solution, covering any constraint that remains undercovered. This process takes roughly as much time as solving the linear relaxation of the problem (approximately 5 seconds in our tests).

The second method of creating initial solution simulates the process through which GISUD^{dis} could be used in a reoptimization context, for instance when reoptimizing an aircrew schedule following an airport closure. In that case, a solution is generally available but no longer optimal. To simulate this

use case, we create perturbed solution from a near optimal solution. The perturbation procedure is the same as in Barry et al. (2024): some variables of the near-optimal solution are swapped with other variables, which are then given a very high cost. Perturbed instances are classified by the fraction of variables that are swapped out of the near optimal solution. In this paper, we consider perturbations of 5%, 10%, and 15% for the main results (Section 6.4). Four initial solutions are created for each instance and each degree of perturbation, yielding a total of $8 \times 3 \times 4 = 96$ instance/initial solution combinations. In Appendix C, we also consider perturbations up to 40%.

6.3 Results using from scratch initial solutions

Table 2 presents the results of GISUD^{dis} and CPLEX for each instance, starting from an initial solution created using the *from scratch* procedure described in Section 6.2. Table 2 first presents the name of the instance, the initial solution cost. For both CPLEX and GISUD^{dis}, we report the objective value of the best solution found (with the relative gap between the solution and the best bound in parenthesis), as well as the resolution time, in seconds. For CPLEX, we also report the best bound computed. For GISUD^{dis}, we also report the maximum incompatibility degree reached ("P m"), as well as a breakdown of the iterations according to which problems were solved: "N its" is the number of iterations, "D N" the number of times the disaggregation procedure is called, with its success rate in parenthesis, and "Z N" the number of times the ZOOM procedure is called.

Table 2: Results GISUD for from scratch tests

Name	Initial Sol	CPLEX			GISUD^{dis}						
		Best bound	Obj. value	Time (s)	Obj. value	Time (s)	P m	N its	D N	Z N
320_1	4138180	1001310	1002650 (0.13)	274	1005451 (0.41)	108	5	28	3 (67%)	6
320_2	4390256	1048110	1049960 (0.18)	314	1052831 (0.45)	107	4	27	10 (40%)	7
320_3	4391685	1257930	1263630 (0.45)	731	1263473 (0.44)	114	4	18	3 (100%)	5
320_4	4130635	1374860	1379690 (0.35)	3074	1381369 (0.47)	299	5	40	8 (38%)	16
757_1	4048506	907796	908928 (0.12)	470	912322 (0.50)	99	5	31	5 (20%)	8
757_2	4580539	938580	941137 (0.27)	3632	942973 (0.47)	144	5	42	17 (53%)	11
757_3	4617367	1178980	1182800 (0.32)	5027	1183559 (0.39)	112	4	26	3 (100%)	3
757_4	7421654	1187680	1191170 (0.29)	5236	1192366 (0.39)	177	4	38	21 (57%)	11
average				2345		145	4	31	8.75 (51%)	8

We observe that for all instances, GISUD^{dis} is between 2.5 and 45 times faster than CPLEX, and 16 times faster on average. The success of GISUD^{dis} is in part due to the CP's ability to quickly find integer direction. Indeed, only (8+8.75*51%)/31 = 40% of iterations on average needed to use either the zoom or disaggregation procedure to conclude. This means that 60% of iterations are performed without resorting to disaggregation or the ZOOM procedure. For such iterations, it is either the RP or the CP that allows to find a better integer solution than the current solution. This better integer solution is then necessarily adjacent to the current solution. Although the GSPP does not have the quasi-integral property, the fact that the right-hand sides are low (less than 4, close to 1) means that the GSPP still remains close to the quasi-integrality property.

A big part of GISUD^{dis}' success on these 60% of iterations is due to the multi-phase strategy. In fact, the maximal incompatibility degree necessary for the resolution of any instance is 5, which is low. Low levels of incompatibility degree greatly favor integral directions. For this reason, the majority of the GISUD iterations are done without the usage of either disaggregation or zoom: this majority is done by obtaining integer solutions from binary compatible columns or from CP resolution in low phases.

For the remaining iterations, on average $8.75 \times 51\%/31 \approx 14\%$ terminate with the disaggregation procedure and $8/31 \approx 25\%$ with the ZOOM procedure.

The success rate of the disaggregation component is very high, averaging 51%. When the disaggregation component is called, half the time it manages to obtain an integer solution of better cost (which could not be found with CP without the disaggregation component). Iterations due to disaggregation represent half of the ZOOM iterations. Calling task disaggregation then avoids a considerable number of calls to the ZOOM procedure. Task disaggregation is based on solving a linear program, whereas the ZOOM procedure is based on solving a certain number of small integer programs (for which the resolution time can be long and difficult to predict). Task disaggregation is therefore faster and more reliable than the ZOOM procedure. It allows optimization to be carried out considerably more quickly.

Notice that CPLEX produces slightly better solutions than GISUD^{dis}. This is because, as a dual method, CPLEX produces relatively few solutions. The first solution to reach the 0.5% optimality threshold is likely to be well below it. By contrast, GISUD^{dis} is a primal method and thus produces a new feasible solution at each iteration. For this reason, the first solution to reach the threshold is more likely to be close to it. The GISUD^{dis} gap is close to 0.5%.

6.4 Result on reoptimization

For *reoptimization* tests, Tables 3, 4, 5, 6 present the results of GISUD^{dis} and CPLEX with the different initial solutions generated for each instance. The columns "Name" and "Pert" correspond to the name of the instance and the perturbation level of the initial solution. The "Initial Sol" column corresponds to the cost of the initial solution. For CPLEX, we indicate the best lower bound obtained at the end of the resolution, the value of the best solution obtained, as well as the total execution time. For GISUD^{dis}, we indicate the value of the best solution obtained, the computation time, the total number of iterations (N. its), the number of calls to the disaggregation procedure (D N) with the percentage of these calls leading to an integer direction in parentheses and the number of calls to the ZOOM procedure (Z N). The last row of each table contains the averages.

Table 3: Results for instances 320_RHS1

Name	Pert.	Initial Sol	GISUD^{dis}					CPLEX		
			Best bound	Solution Value	Time (s)	Solution Value	Time (s)	N. its	D N	Z N
320_1	5%	1325421	1001310	1003120 (0.18)	270	1005721 (0.44)	**19**	14	0 (0%)	0
320_1	5%	1343078	1001310	1002680 (0.14)	230	1006286 (0.49)	**48**	24	11 (64%)	4
320_1	5%	1301811	1001310	1003060 (0.17)	744	1006199 (0.49)	**10**	9	0 (0%)	0
320_1	5%	1262021	1001310	1003090 (0.18)	401	1005881 (0.45)	**20**	18	0 (0%)	0
320_2	5%	1366909	1048110	1049390 (0.12)	238	1052074 (0.38)	**11**	7	0 (0%)	0
320_2	5%	1372468	1048110	1049700 (0.15)	372	1053301 (0.49)	**11**	9	1 (100%)	0
320_2	5%	1459960	1048110	1049750 (0.16)	288	1052177 (0.39)	**28**	15	1 (100%)	0
320_2	5%	1407292	1048110	1049900 (0.17)	223	1052895 (0.45)	**31**	21	5 (80%)	1
320_1	10%	1556353	1001310	1003470 (0.22)	294	1006040 (0.47)	**51**	25	3 (67%)	1
320_1	10%	1553414	1001310	1002850 (0.15)	297	1005862 (0.45)	**39**	25	3 (67%)	1
320_1	10%	1590620	1001310	1003150 (0.18)	297	1006307 (0.50)	**63**	31	6 (67%)	4
320_1	10%	1542914	1001310	1002570 (0.13)	214	1006098 (0.48)	**55**	26	3 (67%)	1
320_2	10%	1757373	1048110	1049540 (0.14)	244	1052862 (0.45)	**60**	25	4 (25%)	3
320_2	10%	1742464	1048110	1049430 (0.13)	256	1053253 (0.49)	**39**	17	2 (0%)	2
320_2	10%	1740708	1048110	1049690 (0.15)	266	1053218 (0.49)	**48**	29	6 (67%)	2
320_2	10%	1698414	1048110	1049780 (0.16)	271	1052677 (0.43)	**36**	25	5 (80%)	1
320_1	15%	1792813	1001310	1003500 (0.22)	405	1006291 (0.50)	**23**	16	1 (100%)	0
320_1	15%	1870885	1001310	1002800 (0.15)	266	1004671 (0.33)	**116**	43	11 (27%)	9
320_1	15%	1833212	1001330	1002660 (0.13)	745	1005257 (0.39)	**65**	33	4 (50%)	2
320_1	15%	1784373	1001310	1002820 (0.15)	367	1005638 (0.43)	**42**	30	3 (33%)	2
320_2	15%	2161593	1048120	1049770 (0.16)	380	1053065 (0.47)	**48**	28	7 (71%)	2
320_2	15%	2075944	1048110	1049620 (0.14)	863	1053171 (0.48)	**117**	49	10 (40%)	8
320_2	15%	2160881	1048110	1049800 (0.16)	449	1052950 (0.46)	**53**	27	6 (67%)	2
320_2	15%	2112448	1048110	1051040 (0.28)	480	1053303 (0.49)	**118**	59	11 (55%)	5
average		1658890	1024711	1026382 (0.16)	369	1029383 (0.45)	**48**	25.2	4.2 (2.4/4.2, 57%)	2.1

Table 4: Results for instances 757_RHS1

Name	Pert.	Initial Sol	GISUD^{dis}					CPLEX		
			Best bound	Solution Value	Time (s)	Solution Value	Time (s)	N. its	D N	Z N
757_1	5%	1184830	907791	911968 (0.46)	669	911744 (0.43)	**21**	17	0 (0%)	0
757_1	5%	1142776	907796	908812 (0.11)	1279	911148 (0.37)	**18**	12	1 (100%)	0
757_1	5%	1178817	907791	908926 (0.12)	544	911688 (0.43)	**19**	18	2 (100%)	0
757_1	5%	1172680	907799	908539 (0.08)	1231	911751 (0.43)	**39**	20	4 (50%)	2
757_2	5%	1229414	938584	939781 (0.13)	752	942650 (0.43)	**24**	10	0 (0%)	0
757_2	5%	1240043	938583	939652 (0.11)	1847	942904 (0.46)	**27**	14	7 (86%)	1
757_2	5%	1239248	938584	940749 (0.23)	3814	942937 (0.46)	**20**	10	1 (100%)	0
757_2	5%	1247612	938585	940443 (0.20)	3061	943115 (0.48)	**18**	11	1 (100%)	0
757_1	10%	1395146	907813	909138 (0.15)	1213	911704 (0.43)	**30**	21	3 (67%)	1
757_1	10%	1453118	907796	909053 (0.14)	414	912237 (0.49)	**99**	33	10 (40%)	7
757_1	10%	1348551	907791	908769 (0.11)	388	912178 (0.48)	**36**	20	2 (0%)	2
757_1	10%	1416075	907804	908776 (0.11)	1248	911995 (0.46)	**26**	23	3 (100%)	0
757_2	10%	1520798	938581	939762 (0.13)	1738	942293 (0.39)	**24**	9	0 (0%)	0
757_2	10%	1480770	938578	940056 (0.16)	1667	942196 (0.38)	**32**	12	2 (0%)	2
757_2	10%	1521501	938577	941642 (0.33)	851	942931 (0.46)	**93**	32	10 (10%)	9
757_2	10%	1478316	938583	939841 (0.13)	1966	943059 (0.47)	**17**	7	0 (0%)	0
757_1	15%	1676298	907798	908948 (0.13)	1357	912291 (0.49)	**72**	37	5 (60%)	3
757_1	15%	1696243	907802	909168 (0.15)	1326	912077 (0.47)	**45**	24	2 (50%)	1
757_1	15%	1624303	907791	909138 (0.15)	615	912262 (0.49)	**44**	31	1 (100%)	0
757_1	15%	1674775	907795	908958 (0.13)	**346**	912245 (0.49)	940	79	60 (0%)	60
757_2	15%	1817199	938577	940028 (0.15)	3243	942896 (0.46)	**71**	26	4 (25%)	3
757_2	15%	1854577	938585	942907 (0.46)	1782	943279 (0.50)	**79**	32	12 (50%)	6
757_2	15%	1767304	938585	941414 (0.30)	537	942508 (0.42)	**42**	20	1 (100%)	1
757_2	15%	1744725	938583	940244 (0.18)	3971	943276 (0.50)	**73**	25	6 (33%)	4
average		1462713	923189	924863 (0.18)	1494	927390 (0.45)	**80**	22.6	5.7 (1.6/5.7, 28%)	4.2

Table 5: Results for instances 320_RHS2

Name	Pert.	Initial Sol	GISUD^{dis}					CPLEX		
			Best bound	Solution Value	Time (s)	Solution Value	Time (s)	N. its	D N	Z N
320_3	5%	1650533	1257940	1262050 (0.33)	2780	1263913 (0.47)	**143**	18	4 (50%)	3
320_3	5%	1630219	1257950	1263470 (0.44)	2068	1264206 (0.49)	**56**	13	1 (100%)	0
320_3	5%	1607464	1257940	1261520 (0.28)	574	1263905 (0.47)	**67**	15	3 (67%)	2
320_3	5%	1653540	1257940	1259780 (0.15)	678	1264174 (0.49)	**33**	7	0 (0%)	0
320_4	5%	1872569	1374860	1381060 (0.45)	3335	1381614 (0.49)	**54**	14	1 (100%)	0
320_4	5%	1803557	1374860	1380320 (0.40)	3897	1381668 (0.49)	**149**	30	10 (70%)	4
320_4	5%	1854933	1374850	1379030 (0.30)	3501	1381420 (0.48)	**71**	10	0 (0%)	1
320_4	5%	1891390	1374850	1379030 (0.30)	2939	1381302 (0.47)	**67**	19	2 (100%)	0
320_3	10%	1964801	1257940	1259610 (0.13)	1481	1263971 (0.48)	**129**	25	8 (50%)	4
320_3	10%	1964381	1257940	1261390 (0.27)	2061	1263367 (0.43)	**119**	23	1 (100%)	3
320_3	10%	1981199	1257950	1260920 (0.24)	2033	1264149 (0.49)	**149**	33	8 (88%)	3
320_3	10%	2012759	1257940	1261630 (0.29)	2324	1264177 (0.49)	**253**	34	9 (67%)	5
320_4	10%	2220014	1374860	1381090 (0.45)	3421	1381634 (0.49)	**884**	41	18 (56%)	14
320_4	10%	2355761	1374850	1380300 (0.39)	3187	1381624 (0.49)	**2165**	58	35 (43%)	21
320_4	10%	2183140	1374860	1379250 (0.32)	3159	1381629 (0.49)	**97**	24	7 (86%)	2
320_4	10%	2241296	1374860	1378130 (0.24)	3032	1381654 (0.49)	**243**	40	16 (31%)	14
320_3	15%	2439137	1257950	1261680 (0.30)	2396	1263965 (0.48)	**185**	35	5 (80%)	4
320_3	15%	2401735	1257940	1261020 (0.24)	1481	1264151 (0.49)	**209**	47	13 (54%)	8
320_3	15%	2409412	1257950	1263530 (0.44)	2304	1263992 (0.48)	**249**	56	9 (89%)	6
320_3	15%	2381424	1257940	1261530 (0.28)	2050	1263745 (0.46)	**209**	47	5 (0%)	10
320_4	15%	2668104	1374860	1378390 (0.26)	3421	1381308 (0.47)	**181**	36	13 (77%)	4
320_4	15%	2858156	1374860	1378620 (0.27)	3361	1381443 (0.48)	**268**	48	13 (46%)	11
320_4	15%	2858914	1374860	1378970 (0.30)	3310	1381514 (0.48)	**178**	27	3 (33%)	6
320_4	15%	2851997	1374860	1377720 (0.21)	2830	1381592 (0.49)	**209**	41	15 (80%)	5
average		2156518	1316400	1320418 (0.3)	2568	1322754 (0.48)	**265**	30.9	8.3 (4.9/8.3, 59%)	5.4

Table 6: Results for instances 757_RHS2

Name	Pert.	Initial Sol	GISUD^{dis}			CPLEX			
			Best bound	Solution Value	Time (s)	Solution Value	Time (s)	N. its	D N	Z N
757_3	5%	1486786	1178980	1182440 (0.29)	5282	1184654 (0.48)	**73**	17	3 (67%)	1
757_3	5%	1469060	1178970	1182630 (0.31)	4678	1184793 (0.49)	**79**	21	0 (0%)	2
757_3	5%	1493802	1178970	1184100 (0.43)	4629	1184537 (0.47)	**70**	14	4 (75%)	2
757_3	5%	1530907	1178970	1181190 (0.19)	3571	1184687 (0.48)	**865**	26	13 (62%)	5
757_4	5%	1505965	1187680	1190110 (0.20)	4999	1192388 (0.39)	**44**	10	5 (60%)	2
757_4	5%	1527269	1187680	1190450 (0.23)	4899	1193159 (0.46)	**83**	15	7 (0%)	7
757_4	5%	1528409	1187690	1190170 (0.21)	4812	1192766 (0.43)	**61**	23	9 (67%)	3
757_4	5%	1520236	1187690	1190780 (0.26)	5957	1192826 (0.43)	**42**	17	6 (83%)	1
757_3	10%	1785493	1178980	1182210 (0.27)	4263	1184650 (0.48)	**158**	31	10 (80%)	4
757_3	10%	1845185	1178980	1183440 (0.38)	4895	1184709 (0.48)	**164**	37	8 (50%)	6
757_3	10%	1868478	1178980	1183920 (0.42)	5156	1184805 (0.49)	**123**	30	5 (40%)	3
757_3	10%	1851434	1178980	1182870 (0.33)	4748	1184068 (0.43)	**176**	29	1 (100%)	3
757_4	10%	1868169	1187690	1190560 (0.24)	4945	1193049 (0.45)	**88**	26	8 (38%)	5
757_4	10%	1828500	1187680	1191900 (0.35)	5187	1192025 (0.36)	**1738**	53	34 (21%)	28
757_4	10%	1835670	1187690	1191150 (0.29)	5471	1193032 (0.45)	**106**	32	11 (27%)	8
757_4	10%	1779339	1187690	1191910 (0.35)	5267	1193577 (0.49)	**73**	25	6 (83%)	1
757_3	15%	2186764	1178970	1183160 (0.35)	4307	1184754 (0.49)	**322**	49	5 (60%)	11
757_3	15%	2148262	1178970	1181640 (0.23)	3719	1184797 (0.49)	**230**	50	19 (63%)	9
757_3	15%	2057260	1178970	1183650 (0.40)	5086	1184829 (0.49)	**163**	43	11 (64%)	6
757_3	15%	2086605	1178980	1181430 (0.21)	4204	1184851 (0.50)	**164**	40	8 (62%)	5
757_4	15%	2128890	1187690	1189400 (0.14)	3324	1193224 (0.46)	**119**	40	7 (86%)	3
757_4	15%	2201273	1187690	1190650 (0.25)	5186	1193417 (0.48)	**147**	41	12 (42%)	8
757_4	15%	2189536	1187690	1191570 (0.33)	5194	1193520 (0.49)	**71**	28	7 (71%)	2
757_4	15%	2211074	1187680	1190330 (0.22)	4744	1193235 (0.47)	**114**	43	9 (78%)	2
average		1830598	1183330	1186735 (0.29)	4772	1188848 (0.46)	**220**	30.8	8.7 (4.6/8.7, 53%)	5.3

For *re-opt* tests, from the result tables, we observe that for instances 320 and 757 with the RHS1 set of right-hand sides, GISUD^{dis} is faster than CPLEX for almost all instances (47/48). CPLEX takes an average of 369s on instances 320_RHS1 and 1494s on instances 757_RHS1. GISUD^{dis}, on the other hand, is more stable on these two groups of instances, taking an average of 48s for instances of type 320_RHS1 and 80s for instances of type 757_RHS1. CPLEX seems much faster on the 320_RHS1 instance group compared to its performance on the 757_RHS1 group. These two instance groups are similar in terms of density and size, and CPLEX preprocessing works similarly on both groups of instances. By examining the CPLEX outputs in more detail, we observe that, on the instances of type 757_RHS1, the branch-and-bound tree contains twice as many nodes as on 320_RHS1. The number of iterations of GISUD^{dis}, however, is similar on 320_RHS1 and 757_RHS1. Solving large problems quickly with the branch-and-bound method is conditioned by the small size of the branch-and-bound tree. However, the size of this branch-and-bound tree is difficult to predict in advance. GISUD^{dis}, as a primal method, appears more stable than CPLEX.

In the case of instances 320_RHS2 and 757_RHS2, GISUD^{dis} is approximately 10 times faster than CPLEX across all instances. GISUD^{dis} takes a similar average time on both sets of instances, while CPLEX takes about twice as long on 757_RHS2 as on 320_RHS2. The average times for GISUD^{dis} and CPLEX are longer on RHS2-type instances than on RHS1-type instances. This indicates an increase in problem difficulty with the value of the right-hand sides.

Regarding the gaps achieved compared to the best lower bound, those of CPLEX are generally smaller than those of GISUD^{dis}, for the reasons mentionned above.

Table 7 summarizes the results of Tables 3–6. Each column corresponds to a group of instances. The "Avg. Time (s)" line corresponds to the average time of CPLEX and GISUD^{dis} on each table. The lines "Avg. Time 5% (s)", "Avg. Time 10% (s)", and "Avg. Time 15% (s)" correspond to the average times of CPLEX and GISUD^{dis} on each table for the different levels of perturbation. The "Success D" line corresponds to the success percentage of the disaggregation component. The line "Iterations

N;D;Z" corresponds to the distribution of the total number of iterations on each table between classical iterations, iterations that succeed thanks to disaggregation, and iterations that succeed thanks to the ZOOM procedure.

Table 7: Statistics on the different test groups

	320_RHS1		320_RHS2		757_RHS1		757_RHS2	
	CPLEX	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**
Avg. Time (s)	369	48	2568	265	1494	80	4772	220
Avg. Time 5% (s)	345	22	2471	80	1649	23	4853	164
Avg. Time 10% (s)	267	48	2587	504	1185	44	4991	328
Avg. Time 15% (s)	494	72	2644	211	1647	170	4470	166
Success D	N/A	56%	N/A	59%	N/A	28%	N/A	53%
Iterations N;D;Z	N/A	498;57;50	N/A	494;117;130	N/A	403;38;102	N/A	503;110;127

In terms of average times, we observe that on 320_RHS1, GISUD^{dis} is 7.6 times faster than CPLEX. On the instance 757_RHS1, GISUD^{dis} is more than 10 times faster than CPLEX. On RHS2-type instances, GISUD^{dis} is also more than 10 times faster than CPLEX in re-optimization. The lower the perturbation percentage, the faster GISUD^{dis}. This is verified on instances 320_RHS1 and 757_RHS1, but less so on RHS2-type instances, mainly due to some outliers that can increase the averages. This shows that GISUD^{dis} benefits from the quality of the initial point in terms of primal information. The gap with CPLEX is therefore even greater when the initial solution is good because CPLEX does not seem to take advantage of the low perturbation percentage of the initial solution (dual method).

For the disaggregation component, we observe that it works similarly on the different levels of perturbation except 757_RHS1. When 757_RHS1 is excluded, about 55% of task disaggregations return one or more integer directions directly. This is why we observe that on these instances, there are about as many iterations achieved thanks to task disaggregation as thanks to ZOOM. On the instance 757_RHS1, the disaggregation component returns one or more integer directions directly in 28% of cases. This is due to a single instance where 60 disaggregations are performed, none of which succeed due to some difficult optimization towards the end (instance 757_RHS1 with 15% perturbation). When this instance is excluded, the results are similar to those of other instances (49% success rate). The success rate of the disaggregation component thus seems stable. This success rate is very high. In half of all cases, applying the disaggregation procedure results in one or more integral improvement directions. There are even several instances where optimization is performed without any call to the ZOOM procedure thanks to this disaggregation component. To evaluate disaggregation in other circumstances, we test with higher levels of perturbations (to 20%, 30%, and 40%). The detailed results are available in the appendix in Tables C1, C2, C3, C4, and a summary is presented in Table 8.

Table 8: Statistics on different test groups with high perturbation

	320_RHS1		320_RHS2		757_RHS1		757_RHS2	
	CPLEX	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**	**CPLEX**	**GISUD^{dis}**
Avg. Time (s)	325	182	2343	518	1667	186	4837	354
Avg. Time 20% (s)	332	102	2067	605	1639	183	4890	267
Avg. Time 30% (s)	303	214	2592	407	2065	155	4654	331
Avg. Time 40% (s)	339	229	2368	539	1294	218	4966	462
Success D	N/A	48%	N/A	48%	N/A	48%	N/A	54%
Iterations N;D;Z	N/A	1116;138;226	N/A	973;213;454	N/A	960;164;244	N/A	1118;267;390

Across the 4 instance groups, GISUD^{dis} has a better average time than CPLEX (up to 10x speedup). The average times of GISUD^{dis} generally increase quite slowly with the percentage of perturbation, while CPLEX is less sensitive to it. The disaggregation component seems to be as successful on highly perturbed instances as on less perturbed ones. On average, it returns one or more integer solutions directly half the times it is called.

6.5 Size of disaggregated problems

The previous subsections show that disaggregation speeds up the resolution. In fact, task disaggregation is simply based on solving a linear program. Moreover the size of this linear program actually remains close to that of the original problem thanks to the phase filter.

Figure 4 shows a histogram of the percentage quotients between the size of the disaggregated problem (number of non-zero coefficients in the constraint matrix of CP_D) and the size of the initial problem (number of non-zero coefficients in the constraint matrix of CP) for the instance group 320_RHS2. On average, the disaggregated complementary problem is 1.15 times larger than the initial complementary problem, and many disaggregated problems are roughly the same size as the initial problem. In our tests, the size of the disaggregated problem was rarely more than 1.5 times and never more than 2 times larger than the original problem.

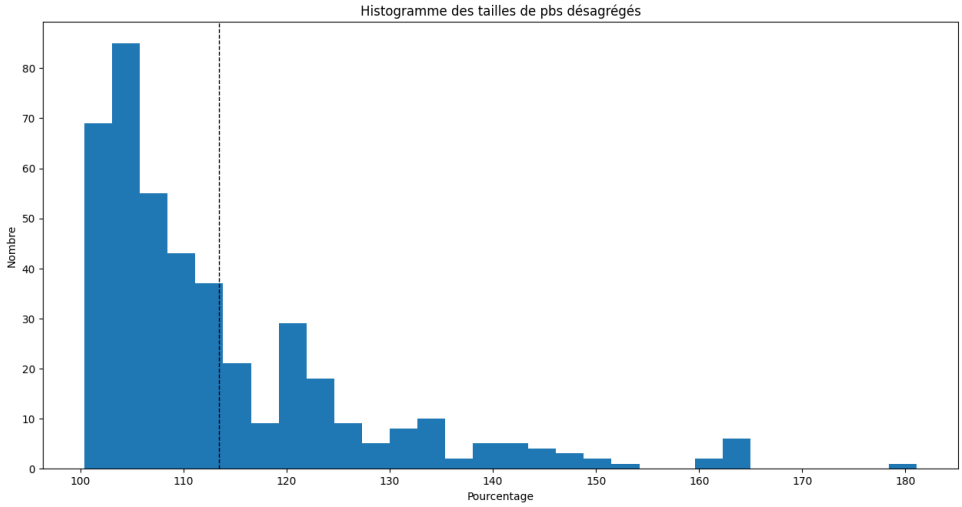

Figure 4: Size of Disaggregated Problems

7 Conclusion

In this work, we have presented a strategy that speeds up the GISUD algorithm. It consists of a component that disaggregates the tasks covered by the CP solution when CP produces a fractional direction. This task disaggregation often (in half the cases) results in an improving integer direction. This speeds up the resolution algorithm: When the strategy succeeds, there is no need to solve integer programs with the ZOOM procedure. This last point is demonstrated by practical results, showing the utility of this component from-scratch and in the context of re-optimization. This work helps to understand that there are, in some cases, alternatives to branching.

Appendix

A Algorithm that permits to compute incompatibility degree

In the incompatibility degree graph of column A_k with respect to $\mathcal{P}$, each arc is assigned with a length: 0, 1, or 2, according to this algorithm, which computes the length L of the arc between node (w_i, A_j) and node $(w_{i'}, A_{j'})$. This algorithm is chosen in such a way that the length of a path in the graph, associated with an assignment, is an upper bound on the size of the slicing $\eta(f)$ associated with the assignment f.

The initial length is $L = 0$. If $succ(i, j) \neq NIL$ and ($j \neq j'$ or $succ(i, j) \neq i'$), L is increased by 1. If $pred(i', j') \neq NIL$ and ($j \neq j'$ or $pred(i', j') \neq i$), L is increased by 1. Arcs from the source to the first stage have a length of 0 if the node at the end represented by (w_i, A_j) has column j starting with task i and 1 otherwise. Arcs from the last stage to the sink have a length of 0 if the node (w_i, A_j) at the end has column j ending with task i and 1 otherwise. The shortest path between source and sink then gives an upper bound on a column's incompatibility measure (column incompatibility degree).

In Section 5.3, for generating the columns of the disaggregated problem $GSPP_D$ of phase smaller than P associated with column A_k illustrated by the example in that section, it is possible to use the incompatibility degree graph of column A_k with respect to $\mathcal{P}$.

Each column of $S_D^{-1}[\{A_k\}]$ covers exactly once the tasks associated with the same initial task (because A_k covers each of the tasks). The number of nodes associated with task i in the graph corresponds exactly to the coverage number of task i and therefore to the number of tasks associated with task i in the disaggregated problem. Each column of $S_D^{-1}[\{A_k\}]$ can therefore be represented by a path in the graph: each node in this graph actually represents a task in the disaggregated problem. The node i, j associated with task i and column j of $\mathcal{P}$ represents the task of $[\![1, m_D]\!]$ equivalent to i (in $\tilde{W}_D(i)$) covered by column $f(j)$ of the disaggregated problem, which is associated with column j of $\mathcal{P}$. The length of the path is exactly equal to our approximation of the degree of incompatibility of the column of $S_D^{-1}[\{A_k\}]$ because the order of the tasks is not changed during disaggregation. It is therefore possible to enumerate all paths shorter than P directly in this graph in order to generate columns with a degree of incompatibility less than P. The length of the paths corresponds to the incompatibility degree of the columns.

B Lemma section 4.2

Lemma 1. Suppose that for $k \in \mathbb{N}^*$ and $j_1, ..., j_{k+1} \in [\![1, n]\!]$, we have $\sum_{l=1}^{k+1} A_{j_l} \leq b$. Suppose also that for each $l \in [\![1, k]\!]$, we have $f(j_l) \in [\![1, n_D^{j_l}]\!]$ chosen from the columns of the $GSPP_D^{dis}$ problem associated with the column A_{j_l}, such that $v^{(k)} = \sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)} \leq \tilde{b}_D$. Then, for each task $i \in \{1, m\}$ covered by column $A_{j_{k+1}}$, there exists $i' \in \tilde{W}_D(i)$, a task associated with task i in the disaggregated problem $GSPP_D^{dis}$, such that $v_{i'}^{(k)} = (\sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)})_{i'} < (\tilde{b}_D)_{i'}$.

Proof. Suppose, on the contrary, that for a task $i \in [\![1, m]\!]$ covered by the column $A_{j_{k+1}}$, we have that for every task $i' \in \tilde{W}_D(i)$, $v_{i'}^{(k)} = (\sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)})_{i'} \geq (\tilde{b}_D)_{i'}$. Now, since for every $l \in [\![1, n]\!]$ and $s \in [\![1, n_D^l]\!]$, $S_D((M_D^l)_s) = A_l$, we have that $S_D(\sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)})_i = \sum_{l=1}^{k} S_D((M_D^{j_l})_{f(j_l)})_i = \sum_{l=1}^{k} (A_{j_l})_i$. Furthermore, we have $S_D(\sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)})_i = \sum_{i'' \in \tilde{W}_D(i)} \left(\sum_{l=1}^{k} (M_D^{j_l})_{f(j_l)} \right)_{i''} \geq \sum_{i'' \in \tilde{W}_D(i)} (\tilde{b}_D)_{i''} = b_i$. We would therefore have $\sum_{l=1}^{k} (A_{j_l})_i \geq b_i$ and thus $\sum_{l=1}^{k+1} (A_{j_l})_i > b_i$ (since the column $A_{j_{k+1}}$ covers the task i), whereas we have $\sum_{l=1}^{k+1} (A_{j_l})_i \leq b_i$. We therefore conclude that there exists a task $i' \in \tilde{W}_D(i)$ such that $v_{i'}^{(k)} < (\tilde{b}_D)_{i'}$. □

C Completion of results

Table C1: Results for 320_RHS1 instances

Name	Pert.	Initial Sol	Best cplex bound	CPLEX solution	CPLEX time	GISUD solution	GISUD time	N its	Z N	D N
320_1	20%	2164571	1001310	1002560 (0.12)	163	1005521 (0.42)	**141**	43	5	8 (38%)
320_1	20%	2145016	1001310	1003090 (0.18)	327	1005650 (0.43)	**98**	44	6	10 (40%)
320_1	20%	2155811	1001320	1003080 (0.18)	408	1005980 (0.46)	**103**	41	6	7 (43%)
320_1	20%	2136545	1001310	1003230 (0.19)	400	1006304 (0.50)	**98**	45	3	6 (67%)
320_2	20%	2509859	1048110	1049720 (0.15)	382	1053335 (0.50)	**75**	41	1	5 (80%)
320_2	20%	2556207	1048110	1049880 (0.17)	373	1053204 (0.48)	**133**	53	10	14 (29%)
320_2	20%	2507792	1048110	1049680 (0.15)	310	1053334 (0.50)	**117**	54	9	18 (50%)
320_2	20%	2541419	1048110	1049490 (0.13)	294	1053021 (0.47)	**51**	30	0	1 (100%)
320_1	30%	2738968	1001310	1002700 (0.14)	**245**	1006118 (0.48)	585	76	18	21 (33%)
320_1	30%	2778317	1001310	1004460 (0.31)	396	1005971 (0.46)	**190**	65	11	8 (12%)
320_1	30%	2662210	1001310	1002980 (0.17)	261	1006263 (0.49)	**143**	60	7	9 (44%)
320_1	30%	2737931	1001310	1003000 (0.17)	182	1005919 (0.46)	**158**	50	9	13 (54%)
320_2	30%	3287443	1048110	1049290 (0.11)	317	1052469 (0.41)	**226**	70	11	5 (60%)
320_2	30%	3116267	1048110	1049590 (0.14)	260	1053345 (0.50)	**119**	53	5	7 (57%)
320_2	30%	3304669	1048110	1049660 (0.15)	400	1052816 (0.45)	**136**	52	7	11 (73%)
320_2	30%	3255856	1048110	1049680 (0.15)	367	1053220 (0.49)	**156**	62	9	15 (53%)
320_1	40%	3451310	1001310	1003160 (0.18)	407	1006048 (0.47)	**243**	83	16	17 (41%)
320_1	40%	3370066	1001310	1002710 (0.14)	253	1006114 (0.48)	**181**	77	11	12 (33%)
320_1	40%	3365495	1001310	1003290 (0.20)	304	1005224 (0.39)	**235**	89	14	10 (50%)
320_1	40%	3421003	1001310	1002780 (0.15)	332	1006240 (0.49)	**257**	87	13	18 (61%)
320_2	40%	3979984	1048110	1049540 (0.14)	307	1051951 (0.37)	**176**	58	8	12 (67%)
320_2	40%	3963125	1048110	1049540 (0.14)	**199**	1053163 (0.48)	349	96	25	32 (44%)
320_2	40%	4061743	1048110	1050210 (0.20)	465	1053119 (0.48)	**219**	86	15	15 (53%)
320_2	40%	4046485	1048110	1049940 (0.17)	447	1053136 (0.48)	**175**	65	7	12 (58%)
		3010753	1024710	1026385 (0.16)	325	1029477 (0.46)	**182**	61.7	9.4	11.9 (5.8/11.9)

Table C2: Results for 320_RHS2 instances

Name	Pert.	Initial Sol	Best cplex bound	CPLEX solution	CPLEX time	GISUD solution	GISUD time	N its	Z N	D N
320_3	20%	2745028	1257940	1259840 (0.15)	566	1264231 (0.50)	**184**	43	8	12 (42%)
320_3	20%	2790158	1257930	1261510 (0.28)	762	1264235 (0.50)	**259**	56	13	18 (44%)
320_3	20%	2644123	1257940	1262580 (0.37)	2488	1263965 (0.48)	**313**	54	11	10 (60%)
320_3	20%	2632857	1257940	1261670 (0.30)	2492	1263925 (0.47)	**1078**	80	48	41 (0%)
320_4	20%	3292091	1374860	1379110 (0.31)	3699	1379943 (0.37)	**406**	46	10	5 (80%)
320_4	20%	3233742	1374860	1380810 (0.43)	3512	1381725 (0.50)	**320**	50	14	14 (36%)
320_4	20%	3277190	1374860	1377930 (0.22)	**416**	1381482 (0.48)	1986	60	17	25 (36%)
320_4	20%	3199304	1374850	1377550 (0.20)	2607	1381226 (0.46)	**299**	53	15	20 (50%)
320_3	30%	3499812	1257940	1262640 (0.37)	974	1264019 (0.48)	**425**	78	21	20 (55%)
320_3	30%	3411146	1257950	1259370 (0.11)	1027	1264150 (0.49)	**322**	60	9	10 (100%)
320_3	30%	3401961	1257940	1261670 (0.30)	2042	1262205 (0.34)	**378**	72	16	24 (54%)
320_3	30%	3365347	1257950	1262360 (0.35)	2572	1263547 (0.44)	**386**	67	16	16 (56%)
320_4	30%	4203718	1374860	1380100 (0.38)	3808	1381625 (0.49)	**412**	74	19	14 (43%)
320_4	30%	4268044	1374860	1379290 (0.32)	3576	1381613 (0.49)	**349**	57	16	18 (61%)
320_4	30%	4171022	1374860	1380360 (0.40)	3775	1379742 (0.35)	**517**	77	20	15 (40%)
320_4	30%	4294081	1374860	1377720 (0.21)	2963	1381667 (0.49)	**473**	62	17	18 (56%)
320_3	40%	4258924	1257940	1259030 (0.09)	1111	1264059 (0.48)	**424**	72	17	17 (47%)
320_3	40%	4370687	1257940	1263350 (0.43)	2856	1264181 (0.49)	**431**	79	21	16 (56%)
320_3	40%	4258692	1257940	1259890 (0.15)	722	1260834 (0.23)	**497**	80	21	18 (67%)
320_3	40%	4270003	1257940	1261000 (0.24)	2280	1264224 (0.50)	**447**	79	22	20 (40%)
320_4	40%	5255113	1374850	1380770 (0.43)	3504	1381600 (0.49)	**839**	111	42	31 (55%)
320_4	40%	5056570	1374860	1377560 (0.20)	2869	1381562 (0.49)	**640**	87	24	29 (45%)
320_4	40%	5250279	1374860	1378100 (0.24)	3357	1380394 (0.40)	**471**	66	18	19 (68%)
320_4	40%	5240708	1374860	1376810 (0.14)	2248	1381408 (0.47)	**569**	77	19	15 (67%)
		3849608	1316399	1320042 (0.28)	2343	1322398 (0.45)	**518**	68.3	18.9	18.5 (8.9/18.5)

Table C3: Results for 757_RHS1 instances

Name	Pert.	Initial Sol	Best cplex bound	CPLEX solution	CPLEX time	GISUD solution	GISUD time	N its	Z N	D N
757_1	20%	1909699	907796	908876 (0.12)	**355**	912151 (0.48)	700	50	9	23 (61%)
757_1	20%	1947339	907795	909393 (0.18)	493	911803 (0.44)	**105**	44	4	8 (75%)
757_1	20%	1992396	907799	909402 (0.18)	440	912056 (0.47)	**67**	36	4	5 (20%)
757_1	20%	2008532	907791	908924 (0.12)	375	912261 (0.49)	**156**	60	10	20 (60%)
757_2	20%	2159547	938583	942995 (0.47)	4027	942894 (0.46)	**43**	19	0	1 (100%)
757_2	20%	2065902	938587	940140 (0.17)	1700	943174 (0.49)	**136**	45	11	11 (27%)
757_2	20%	2038708	938585	940411 (0.19)	1746	943214 (0.49)	**71**	31	2	6 (67%)
757_2	20%	2195301	938577	940917 (0.25)	3983	943127 (0.48)	**191**	45	10	10 (50%)
757_1	30%	2417050	907800	908707 (0.10)	389	912225 (0.49)	**143**	59	6	11 (64%)
757_1	30%	2485179	907803	908956 (0.13)	1290	912320 (0.50)	**120**	66	5	9 (56%)
757_1	30%	2453481	907800	908926 (0.12)	541	912312 (0.49)	**97**	47	4	8 (75%)
757_1	30%	2453011	907805	908655 (0.09)	1262	912066 (0.47)	**247**	78	15	21 (52%)
757_2	30%	2787612	938576	940848 (0.24)	3864	943007 (0.47)	**164**	60	12	14 (43%)
757_2	30%	2789456	938585	940632 (0.22)	3614	943185 (0.49)	**171**	52	10	12 (50%)
757_2	30%	2667912	938578	940980 (0.26)	3852	943133 (0.48)	**205**	62	14	21 (52%)
757_2	30%	2614080	938577	939827 (0.13)	1711	943170 (0.49)	**97**	36	4	10 (70%)
757_1	40%	2931914	907791	909469 (0.18)	611	912116 (0.47)	**230**	78	18	21 (48%)
757_1	40%	2964752	907804	908993 (0.13)	415	911883 (0.45)	**176**	67	10	11 (27%)
757_1	40%	2931712	907795	909059 (0.14)	593	912307 (0.49)	**164**	66	13	15 (33%)
757_1	40%	3022481	907799	909099 (0.14)	572	912249 (0.49)	**235**	83	18	24 (29%)
757_2	40%	3299400	938577	940386 (0.19)	3619	943007 (0.47)	**255**	84	21	27 (33%)
757_2	40%	3354828	938582	940288 (0.18)	1999	943067 (0.48)	**245**	79	18	21 (33%)
757_2	40%	3351833	938580	939878 (0.14)	772	942420 (0.41)	**187**	50	8	10 (60%)
757_2	40%	3339057	938580	940064 (0.16)	1778	943177 (0.49)	**254**	71	18	20 (60%)
		2590882	923189	924826 (0.18)	1667	927596 (0.48)	**186**	57.0	10.2	14.1 (6.8/14.1)

Table C4: Results for 757_RHS2 instances

Name	Pert.	Initial Sol	Best cplex bound	CPLEX solution	CPLEX time	GISUD solution	GISUD time	N its	Z N	D N
757_3	20%	2530571	1178980	1181970 (0.25)	4563	1184624 (0.48)	**388**	74	18	19 (53%)
757_3	20%	2362958	1178980	1181830 (0.24)	4637	1183587 (0.39)	**253**	59	8	7 (43%)
757_3	20%	2414336	1178970	1181730 (0.23)	4658	1183657 (0.40)	**177**	45	4	4 (75%)
757_3	20%	2543765	1178980	1184800 (0.49)	3963	1184343 (0.45)	**316**	59	14	18 (44%)
757_4	20%	2516674	1187680	1190450 (0.23)	5554	1193390 (0.48)	**369**	76	20	33 (48%)
757_4	20%	2461914	1187690	1190640 (0.25)	5031	1193431 (0.48)	**199**	65	11	19 (53%)
757_4	20%	2512672	1187690	1190860 (0.27)	4946	1192231 (0.38)	**227**	56	12	19 (42%)
757_4	20%	2505038	1187690	1191670 (0.33)	5771	1193193 (0.46)	**212**	53	9	18 (61%)
757_3	30%	3148295	1178970	1183370 (0.37)	5132	1184832 (0.49)	**289**	57	11	8 (88%)
757_3	30%	3102671	1178980	1181200 (0.19)	3912	1184599 (0.47)	**359**	69	14	19 (47%)
757_3	30%	3078211	1178970	1181740 (0.23)	4295	1184867 (0.50)	**307**	78	15	20 (50%)
757_3	30%	3053794	1178980	1183970 (0.42)	4519	1184867 (0.50)	**526**	82	23	20 (50%)
757_4	30%	3069951	1187690	1192520 (0.41)	4581	1192909 (0.44)	**262**	79	10	23 (65%)
757_4	30%	3113682	1187690	1190230 (0.21)	4892	1193140 (0.46)	**329**	85	18	32 (53%)
757_4	30%	3124066	1187680	1190640 (0.25)	4768	1193562 (0.49)	**266**	75	15	27 (56%)
757_4	30%	3074170	1187690	1189050 (0.11)	5137	1193609 (0.50)	**310**	76	18	27 (41%)
757_3	40%	3776229	1178970	1182780 (0.32)	4886	1184251 (0.45)	**581**	83	27	21 (62%)
757_3	40%	3610644	1178970	1182210 (0.27)	5209	1184859 (0.50)	**362**	68	15	17 (53%)
757_3	40%	3658925	1178980	1182720 (0.32)	4468	1182794 (0.32)	**575**	84	20	23 (57%)
757_3	40%	3753432	1178970	1181970 (0.25)	3436	1183324 (0.37)	**517**	95	22	16 (75%)
757_4	40%	3844712	1187680	1191640 (0.33)	5113	1192767 (0.43)	**372**	75	20	17 (41%)
757_4	40%	3946385	1187690	1190750 (0.26)	5238	1192475 (0.40)	**545**	113	23	37 (73%)
757_4	40%	3801382	1187680	1191450 (0.32)	5918	1193244 (0.47)	**351**	83	19	20 (40%)
757_4	40%	3709618	1187690	1190260 (0.22)	5461	1193539 (0.49)	**394**	86	24	32 (47%)
		3113087	1183330	1186685 (0.28)	4837	1188670 (0.45)	**354**	74.0	16.2	20.7 (11.1/20.7)

References

Egon Balas and Manfred W. Padberg. On the set-covering problem. Operations Research, 20(6):1152–1161, 1972. URL http://www.jstor.org/stable/169305.

Alpha-Saliou Barry, Frédéric Quesnel, Issmail El Hallaoui, and François Soumis. Algorithme primal ajoutant des variables pour le problème du partitionnement d'ensemble généralisé. Les Cahiers du GERAD G–2024–11, Groupe d'études et de recherche en analyse des décisions, GERAD, Montréal QC H3T 2A7, Canada, January 2024. URL https://www.gerad.ca/fr/papers/G-2024-11.

Adi Ben-Israel and A Charnes. On some problems of diophantine programming. Cahiers du Centre d'Études de Recherche Opérationnelle, 4:215–280, 1962.

Antoine Deza, Sebastian Pokutta, and Lionel Pournin. The complexity of geometric scaling. Operations Research Letters, 52:107057, 2024. doi: https://doi.org/10.1016/j.orl.2023.11.010. URL https://www.sciencedirect.com/science/article/pii/S0167637723001980.

Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. Journal of global optimization, 6:109–133, 1995.

Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. Annals of Operations Research, 140:189–213, 2005.

Fred Glover. A new foundation for a simplified primal integer programming algorithm. Operations Research, 16(4):727–740, 1968. URL http://www.jstor.org/stable/168295.

Fred Glover. Tabu search—part i. ORSA Journal on computing, 1(3):190–206, 1989.

Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), pages 380–384, 2019. doi: 10.1109/COMITCon.2019.8862255.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. Econometrica, 28(3):497–520, 1960. URL http://www.jstor.org/stable/1910129.

Adam Letchford and Andrea Lodi. Primal cutting plane algorithms revisited. Mathematical Methods of Operational Research, 56:67–81, 11 2002. doi: 10.1007/s001860200200.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. Computers & operations research, 24 (11):1097–1100, 1997.

Samuel Rosat, Issmail Elhallaoui, François Soumis, and Andrea Lodi. Integral simplex using decomposition with primal cutting planes. Math. Program., 166(1–2):327–367, November 2017. doi: 10.1007/s10107-017-1123-x. URL https://doi.org/10.1007/s10107-017-1123-x.

Adil Tahir, Guy Desaulniers, and Issmail El Hallaoui. Integral column generation for the set partitioning problem. EURO Journal on Transportation and Logistics, 8(5):713–744, 2019. doi: https://doi.org/10.1007/s13676-019-00145-6. URL https://www.sciencedirect.com/science/article/pii/S219243762030042X.

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

R. D. Young. A simplified primal (all-integer) integer programming algorithm. Operations Research, 16(4): 750–782, 1968. doi: 10.1287/opre.16.4.750. URL https://doi.org/10.1287/opre.16.4.750.

Richard D Young. A primal (all-integer) integer programming algorithm. J. Res. Nat. Bur. Standards.—1965, B, 69:213–250, 1965.

Abdelouahab Zaghrouti, François Soumis, and Issmail El Hallaoui. Integral simplex using decomposition for the set partitioning problem. Operations Research, 62(2):435–449, 2014. URL http://www.jstor.org/stable/24540586.

Abdelouahab Zaghrouti, Issmail Elhallaoui, and François Soumis. Improving set partitioning problem solutions by zooming around an improving direction. Annals of Operations Research, 284, 01 2020. doi: 10.1007/s10479-018-2868-1.