Les Cahiers du GERAD ISSN: 0711–2440

Algorithme primal ajoutant des variables pour le problème du partitionnement d'ensemble généralisé

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

G–2024–11

January 2024

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, 2024
– Bibliothèque et Archives Canada, 2024

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, 2024
– Library and Archives Canada, 2024

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

Algorithme primal ajoutant des variables pour le problème du partitionnement d'ensemble généralisé

Alpha-Saliou Barry [a, b]

Frédéric Quesnel [b, c]

Issmail El Hallaoui [a, b]

François Soumis [a, b]

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

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

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

alpha-2-saliou.barry@polymtl.ca
frederic.quesnel@polymtl.ca
issmail.elhallaoui@polymtl.ca
francois.soumis@polymtl.ca

January 2024
Les Cahiers du GERAD
G–2024–11

Résumé : Le problème du partitionnement d'ensemble est un problème de programmation en nombres entiers très étudié. Le problème consiste à trouver une partition de tâches de coût minimal parmi un ensemble de parties admissibles. Les algorithmes primaux spécialisés ont montré leur capacité à résoudre ces problèmes rapidement. Ces algorithmes trouvent une séquence améliorante de solutions entières jusqu'à la solution optimale. Dans cet article, nous présentons un algorithme primal pour une généralisation de ce problème, généralisation où certaines taches doivent être couverte plusieurs fois, un nombre de fois déterminé à l'avance. Le graphe des solutions entières, où les arcs sont les déplacements possibles avec des pivots du simplexe, n'est pas connexe. L'idée est d'ajouter des variables qui permettent de se déplacer d'une solution à des solutions qui ne sont pas directement voisines dans le graphe afin de restaurer la connexité. Nous montrons ensuite, qu'en pratique, notre méthode est entre 2 à 5 fois plus rapide que les méthodes traditionnelles (CPLEX) sur de larges instances de problèmes de rotations d'équipage aérien.

Mots clés : Algorithme primal, partitionnement d'ensemble généralisé

1 Introduction

Le *problème du partitionnement d'ensemble* (SPP) est un problème d'optimisation bien connu qui peut être énoncé comme il suit. Considérons des variables $x \in \{0,1\}^n$, un vecteur de coût $c \in \mathbb{R}^n$ et une matrice binaire $A = \{0,1\}^{m \times n}$. Le SPP est formulé comme :

$$\min c^T x \tag{1}$$

$$s.c.$$

$$Ax = \mathbb{1}^m \tag{2}$$

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

Chaque ligne des contraintes du SPP est interprétée comme une tache qui doit être couverte exactement une fois et chaque variable (colonne de la matrice A) couvre certaines taches. L'objectif du SPP est de sélectionner un ensemble de variables qui couvre chaque tache exactement une fois tout en minimisant le coût. Le SPP permet de modéliser beaucoup de problèmes pratiques importants. Les problèmes de rotations d'équipage aérien (CPP) pour les pilotes, les problèmes de construction d'horaires (CRP) et les problèmes de routes de véhicules (VRP) peuvent être formulés comme des SPPs. Il y a donc plusieurs algorithmes spécialisés tels que l'algorithme *integral simplex using decomposition* (ISUD Zaghrouti et al. (2014)) qui ont été proposés pour résoudre le SPP.

Dans ce papier, nous proposons une généralisation du SPP, appelée partitionnement d'ensemble généralisé (GSPP) où certaines taches doivent être couvertes plus d'une fois. Le GSPP est énoncé comme il suit :

$$\min_{x \in \{0,1\}^n} c^T x \tag{4}$$

$$s.c.$$

$$Ax = b \tag{5}$$

$$x \in \{0,1\}^n \tag{6}$$

$b_i, i \in [0, ..., m]$ représente le nombre de fois que la tache i doit être couverte. La résolution du GSPP consiste alors à rechercher un sous-ensemble de variables qui permet de couvrir chaque tache i b_i fois à coût minimal.

Plusieurs problèmes d'optimisation peuvent être modélisés comme des GSPPs. Prenons par exemple le CPP pour le personnel de cabine. Ce problème consiste à trouver un ensemble de *pairings* (séquences de vols, de repos et de correspondances, formant un ou plusieurs jours de travail) qui couvre chaque vol un nombre déterminé de fois. Cependant, les exigences en matière d'équipage varient souvent d'un vol à l'autre, de sorte qu'aucun équipage ne peut travailler ensemble pendant toute la durée d'un pairing. Par exemple, supposons qu'il y ait trois vols f^1, f^2 et f^3 nécessitant respectivement 8, 10 et 12 membres d'équipage. Il est impossible de relier deux de ces vols tout en conservant les équipes intactes. Ce problème est généralement traité dans une étape préliminaire en divisant les besoins en équipage de chaque vol en quelques sous-équipes. Par exemple, on peut envisager des sous-équipes de 4 membres et de 2 membres. Le vol f^1 est alors couvert par deux sous-équipes de 4 membres, f^2 par deux sous-équipes de 4 membres et une sous-équipe de 2 membres, et f^3 par trois sous-équipes de 4 membres. Pour ce faire, la solution de CPP doit couvrir f^1 et f^2 deux fois, et f^3 trois fois. Le problème est donc modélisé comme un GSPP. Nous remarquons que dans la plupart des cas pratiques, les membres droits des contraintes 5 restent petits.

La résolution du GSPP peut s'avérer difficile, en particulier pour les instances de grande taille. Les méthodes traditionnelles de programmation en nombres entiers peinent à trouver une solution de bonne qualité dans des délais raisonnables et les algorithmes spécialisés développés pour le SPP ne peuvent pas être directement utilisés pour résoudre le GSPP. Dans cet article, nous proposons une nouvelle méthode de résolution pour le GSPP, appelée *generalized integral simplex using decomposition*

(GISUD). Bien que GISUD suive les mêmes étapes de base que ISUD, la plupart des structures de données et des procédures clés sont adaptées au GSPP. Cela est logique puisque nous remarquons que dans la plupart des cas pratiques, le membre droit des contraintes (5) reste petit (≤ 4), de sorte que ces instances de GSPP sont très proches de celles d'un SPP. GISUD inclut également une *procédure de rajout de colonne* qui permet d'accélérer la résolution.

Les contributions de ce document peuvent être résumées comme il suit :

— Nous adaptons la méthode ISUD au GSPP. Ceci n'est pas trivial car ISUD s'appuie sur la propriété *quasi-intégralité* trouvée dans le SPP mais pas dans le GSPP. À notre connaissance, GISUD est le premier algorithme proposé pour résoudre spécifiquement cette classe de problèmes. Nous proposons notamment de nouvelles définitions de *compatibilité des colonnes* et de *degré d'incompatibilité*. Nous proposons également une stratégie *de rajout de colonnes* qui accélère la méthode et remédie partiellement à l'absence de la propriété de quasi-intégralité.

— Nous testons GISUD sur plusieurs instances à grande échelle de CPP. Les résultats montrent que notre méthode est entre deux et cinq fois plus rapide que les méthodes conventionnelles de programmation en nombres entiers (CPLEX).

— Nous présentons quelques résultats théoriques sur le GSPP et la méthode GISUD. En particulier, nous dérivons une borne supérieure théorique sur le nombre minimum d'itérations de la méthode GISUD.

Le reste de ce document est structuré comme il suit. Nous présentons une revue de la littérature sur le SPP, le GSPP et les méthodes primales pour les deux problèmes, dans la section 2. Nous rappelons les propriétés du GSPP et introduisons une notation utile dans la section 3. La méthode GISUD est présentée dans la section 4 et les résultats expérimentaux sont discutés dans la section 5. Nous obtenons des résultats théoriques sur l'algorithme GISUD dans la section 6 et nous concluons dans la section 7.

2 Revue de littérature

Le problème de partitionnement d'ensemble généralisé peut être formulé comme un problème de programmation en nombres entiers binaire. Ces problèmes peuvent être formulés comme des programmes linéaires dans lesquels les variables sont binaires. Ces variables binaires sont utilisées pour indiquer la non-divisibilité des quantités concernées. La forme générale d'un problème de programmation en nombres entiers binaire est la suivante.

$$\min c^T x \tag{7}$$
$$s.c\ Ax = b \tag{8}$$
$$x \in \{0,1\}^n \tag{9}$$

Dans la pratique, la méthode branch-and-cut, où une relaxation linéaire de 7–9 est utilisée à chaque nœud d'un arbre de branchement, est souvent employée pour résoudre ce type de problème. Dans le cas particulier de la programmation binaire, les points entiers correspondent tous à des points extrêmes du polytope relaxé. Le SPP correspond au cas où la matrice A est binaire et où b ne comporte que des 1. Le GSPP correspond au cas où la matrice A est binaire mais où b contient des entiers strictement positifs.

2.1 Algorithmes primaux

Les algorithmes branch-and-cut appartiennent à la famille des algorithmes duaux. Ces algorithmes résolvent des relaxations (généralement des relaxations linéaires) du problème d'optimisation, en gardant comme invariant l'optimalité primale et duale de la solution actuelle pour la relaxation. Ces relaxations sont renforcées au fil du temps en ajoutant des coupes et en effectuant des branchements, et l'algorithme s'arrête lorsque l'intégralité est atteinte. À l'inverse, les algorithmes primaux conservent

la faisabilité comme invariant (l'intégralité dans le cas des MIP) et s'efforcent d'atteindre l'optimalité. Cela se fait généralement en générant une succession de solutions sous-optimales jusqu'à ce que l'optimalité soit atteinte. Ces solutions sont générées de manière itérative et leur coût diminue au fur et à mesure que le processus d'optimisation progresse. Les algorithmes primaux exacts pour les programmes en nombres entiers sont apparus pour la première fois en 1962 avec les travaux de Ben-Israel et Charnes (1962). Cet algorithme s'inspire du simplexe primal : il part d'une solution entière de base réalisable et identifie les pivots qui permettent à la solution de rester entière. Dans les cas où de tels pivots ne peuvent être identifiés, un plan coupant est ajouté et un pivot sur un coefficient de ce plan est effectué. Ce pivot a la particularité de préserver l'intégralité. Des variantes ont été développées (Young, 1965, 1968; Glover, 1968). Par la suite, cependant, la recherche s'est principalement concentrée sur l'amélioration des méthodes duales. À partir des années 1990/2000, la recherche a repris sur les algorithmes primaux. Des méthodes de résolution numérique ont été proposées : Firla et al. (2001) et Letchford et Lodi (2002). Un portrait plus complet des algorithmes primaux est fourni par Spille et Weismantel (2005).

La principale caractéristique de l'algorithme branch-and-cut est qu'il peut être utilisé pour résoudre n'importe quel problème de programmation linéaire en nombres entiers. Cette classe de problèmes est très large. Pour une sous-classe de problèmes, il peut être avantageux de développer une méthode spécialisée qui tire parti des propriétés de la classe de problèmes. Il existe peu de littérature sur les méthodes exactes de résolution du GSPP. En revanche, le SPP est bien étudié. Les algorithmes primaux, dont certains sont présentés dans la section 2.3, constituent une famille d'algorithmes qui a montré de bonnes performances dans la résolution rapide du SPP.

Les méta-heuristiques sont d'autres algorithmes qui peuvent être considérés comme des algorithmes primaux. En général, les méta-heuristiques calculent des solutions réalisables et effectuent une recherche locale (procédure de recherche adaptative randomisée gloutone (Feo et al., 1994), recherche locale itérée (Stützle, 1998)) ou une recherche globale (algorithmes génétiques (Holland, 1992)) afin d'améliorer ces solutions. Ces méthodes sont généralement plus rapides à exécuter que les méthodes primales exactes, mais n'offrent aucune possibilité de contrôler (ou même d'évaluer) l'écart d'optimalité. De nombreuses méta-heuristiques existent pour résoudre les problèmes d'optimisation en nombres entiers. Hussain et al. (2019) constitue une revue de la recherche sur les méta-heuristiques entre les années 1983 et 2016.

2.2 Problème du partitionnement d'ensemble

La recherche d'algorithmes primaux exacts pour le SPP a intéressé de nombreux chercheurs depuis 1969. La plupart des approches s'appuient sur la propriété de quasi-intégralité que possède le polyèdre de la relaxation linéaire du SPP. Cette propriété stipule que chaque arête de l'enveloppe convexe des points entiers du SPP est également une arête du polyèdre de relaxation linéaire (Trubin (1969)). En utilisant cette propriété, Balas et Padberg (1972) montrent qu'au plus m pivots (où m est le nombre de contraintes de partitionnement) sont nécessaires pour atteindre l'optimalité à partir de n'importe quelle solution entière initiale. Cette séquence de pivots correspond à un chemin le long des arêtes du polyèdre de la relaxation visitant une séquence de points entiers. De plus, le coût des solutions visitées forme une séquence monotone (décroissante dans le contexte de la minimisation). Cependant, trouver un tel chemin peut s'avérer difficile en pratique. La quasi-intégralité est particulièrement intéressante car elle implique qu'il est possible d'améliorer une solution entière en effectuant une recherche locale d'un pivot conduisant à une solution entière améliorée. Le théorème Balas et Padberg (1972) a inspiré de nombreux algorithmes primaux efficaces pour le SPP, dont certains sont expliqués dans la sous-section 2.3.

Dans de nombreux cas, les colonnes de la matrice du SPP ne sont pas toutes connues à l'avance. Elles sont alors générées dynamiquement à l'aide de l'algorithme de génération de colonnes. La génération de colonnes permet de résoudre des problèmes de programmation linéaire à grande échelle en ne considérant qu'un sous-ensemble de colonnes du problème dans un *problème maître* et en intégrant

progressivement davantage de colonnes par la résolution de *sous-problèmes*. Ces sous-problèmes sont souvent formulés comme des problèmes de plus court chemin avec contraintes de ressources dans un graphe. De nombreux problèmes pratiques sont résolus à l'aide de la méthode branch-and-price, qui est une méthode duale inspirée de la méthode branch-and-bound : la différence est qu'à chaque nœud, la relaxation linéaire est résolue à l'aide de la génération de colonnes (Barnhart et al., 1998). Des développements récents (par exemple, Rönnberg et Larsson (2009); Tahir et al. (2019)) ont également comblé le fossé entre les méthodes primales exactes pour le SPP et le branch-and-price. Dans ces méthodes, le problème maître est formulé comme un SPP (au lieu d'une relaxation linéaire de SPP comme c'est généralement le cas) et est résolu à l'aide d'un algorithme primal. Les colonnes sont ajoutées à ce problème maître au fur et à mesure qu'elles apparaissent.

2.3 Algorithmes primaux pour le SPP

Certains algorithmes primaux ont été développés spécifiquement pour le SPP statique (avec la matrice A connue à l'avance). Les algorithmes primaux développés par Haus et al. (2001) et Thompson (2002) en sont des exemples. À partir d'une solution initiale, une première étape identifie les pivots entiers non dégénérés jusqu'à atteindre un optimum local. Ensuite, différentes bases dégénérées associées à cet optimum local sont explorées via un arbre de branchement jusqu'à ce qu'un pivot entier améliorant la solution soit trouvé. Ces méthodes sont combinatoires et souffrent donc de dégénérescence : de nombreuses bases peuvent être explorées avant qu'un pivot améliorant ne soit trouvé. L'un des algorithmes les plus prometteurs disponibles aujourd'hui pour résoudre le SPP est l'algorithme Integral simplex using decomposition (ISUD) proposé par Zaghrouti et al. (2014). ISUD résout partiellement le problème de dégénérescence en adoptant une vision géométrique de l'adjacence : il prend en compte les arêtes adjacentes au point courant, plutôt que les pivots, éliminant ainsi la dégénérescence. Ce travail s'inspire des recherches de Pan (1998) et de l'algorithme d'agrégation dynamique de contraintes (Elhallaoui et al. (2005)). Dans la méthode ISUD, toute solution entière est représentée de manière unique par une *base de travail* qui ne contient que des variables de base ayant une valeur strictement positive. À partir d'une solution entière, le problème de la recherche d'une meilleure solution entière est décomposé en deux sous-problèmes : le problème réduit (RP) et le problème complémentaire (CP). Dans le RP, seuls les pivots non dégénérés menant à une meilleure solution entière sont pris en compte. Lorsque l'amélioration n'est plus possible dans le RP, le CP est résolu pour identifier l'arête du polyèdre dont la direction de support normalisée a le coût réduit le plus faible. Dans la pratique, il a été observé que cette arête conduit souvent à une meilleure solution entière, et si ce n'est pas le cas, un branchement peut être effectué pour en trouver une. ISUD est plus performant que CPLEX pour les gros problèmes (Zaghrouti et al. (2014)). Une approche en plusieurs phases, qui donne la priorité aux colonnes les moins incompatibles dans le CP, accélère la résolution.

Plusieurs améliorations d'ISUD ont été proposées. L'une d'entre elles consiste à modifier les poids de la contrainte de normalisation du CP (une contrainte qui borne le problème, Rosat et al. (2017)). Zaghrouti et al. (2020) proposent de traiter l'intégralité dans le RP au lieu du CP. D'autres améliorations étendent le voisinage de recherche aux solutions qui ne sont pas directement adjacentes à la solution actuelle. Pour ce faire, toutes les colonnes d'une solution sont regroupées en une seule colonne artificielle qui couvre toutes les tâches. On peut montrer que cette colonne artificielle représente un point artificiel qui est adjacent à tous les points extrêmes entiers. Il est donc possible, en recherchant une nouvelle solution adjacente à ce nouveau point artificiel, d'obtenir des solutions entières non directement adjacentes au point actuel. Les objectifs de ces stratégies sont soit d'accélérer la résolution (Zaghrouti et al. (2018)), soit de généraliser la méthode à des variantes du SPP qui n'ont pas la propriété de quasi-intégralité (Tahir et al. (2022)). La stratégie de rajout de colonnes présentée à la section 4.3 s'inspire de cette idée.

3 Propriétés du GSPP

Cette section présente certaines propriétés du GSPP qui sont utiles pour comprendre le fonctionnement de l'algorithme GISUD . Nous rappelons quelques propriétés algébriques et géométriques des programmes linéaires, dans le contexte du GSPP.

Soit $GSPP_r$ la relaxation linéaire du GSPP. Soit X l'ensemble des points extrêmes de $GSPP_r$ et $Conv(X) = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ l'enveloppe convexe de ses points extrêmes (tout au long de ce document, $Conv(\cdot)$ désigne l'enveloppe convexe de l'ensemble des points $(\cdot)$). Algébriquement, $Conv(X)$ est l'ensemble des solutions positives d'une équation matricielle. Géométriquement, c'est l'ensemble des points d'un polyèdre convexe. Nous rappelons ici les liens entre ces deux points de vue. Dans la suite de cet article, nous supposons que la matrice A ne contient pas de colonnes nulles.

3.1 Point de vue algébrique

D'un point de vue algébrique, toute solution $x \in Conv(X)$ peut être exprimée comme une combinaison linéaire de *solutions extrêmes*, où X est l'ensemble des solutions extrêmes. Les solutions extrêmes, à leur tour, peuvent être exprimées en termes de *base*. La notion de *base généralisée*, tirée de l'algorithme du simplexe à variables bornées, présente un intérêt particulier pour le GSPP :

Définition 1. (Base généralisée) Une solution extrême $x \in X$ peut être représentée en partitionnant les variables en trois ensembles (B_x, L_x, U_x) tels que les variables dans B_x forment une base de A et que $\{x_i | 0 < x_i < 1\} \subseteq B_x$, $L_x = \{x_i | x_i = 0\} \setminus B_x$, et $U_x = \{x_i | x_i = 1\} \setminus B_x$. (B_x, L_x, U_x) est appelée base généralisée.

La base généralisée (B_x, L_x, U_x) est créée en incluant toutes les variables libres dans B_x, en plus de quelques variables à leur borne inférieure et supérieure afin de compléter la base. Les autres variables sont incluses dans L_x ou U_x selon qu'elles se trouvent à leur borne inférieure ou supérieure dans x. Une propriété bien connue de la programmation linéaire est que si $Conv(X) \neq \emptyset$, alors au moins une base généralisée correspond à une solution optimale.

Notons que lorsque x contient moins de m variables libres, plusieurs bases généralisées correspondent à ce point extrême. La solution x est alors dite *dégénérée*. En particulier, si $x \in X$ est entier, toutes les variables sont à leur borne inférieure ou supérieure, donc *n'importe quelle* sélection de m variables peut former B_x dans une base généralisée associée à x.

La notation de la base généralisée est utile car elle permet d'utiliser des outils d'algèbre linéaire pour mettre en relation différents points extrêmes. En particulier, un changement de solution peut être exprimé en termes de pivots du simplexe. Un pivot est effectué lorsqu'une variable non basique x_i entre dans la base et qu'une variable basique correspondante x_j en sort. Deux types de pivots peuvent être effectués dans l'algorithme du simplexe avec des variables bornées :

Pivot dégénéré : x_i entre dans la base sans changer de valeur. Dans ce cas, la variable sortante x_j se trouve à sa borne inférieure ou supérieure et est ajoutée en conséquence à L_x ou U_x. La base généralisée résultante correspond au même point extrême.

Pivot non dégénéré : La valeur de x_i change en entrant dans la base. Dans ce cas, la valeur de la variable sortante x_j atteint sa borne supérieure ou inférieure et est ajoutée à L_x ou U_x, respectivement.

Les pivots non dégénérés sont les plus intéressants car ils permettent de passer d'un point extrême à un autre différent.

Comme dans la programmation linéaire standard, nous définissons l'adjacence entre les bases généralisées.

Définition 1. (Bases adjacentes) Considérons deux bases généralisées $(B_{x^1}, L_{x^1}, U_{x^1})$ et $(B_{x^2}, L_{x^2}, U_{x^2})$ associées aux points extrêmes x^1 et x^2, respectivement (x^1 peut être égal à x^2). Les deux bases

généralisées sont adjacentes l'une à l'autre si et seulement s'il est possible d'obtenir $(B_{x^1}, L_{x^1}, U_{x^1})$ en effectuant un seul pivot (dégénéré ou non) sur $(B_{x^2}, L_{x^2}, U_{x^2})$.

Pour le GSPP, l'exécution de pivots conduisant à une solution différente, meilleure, peut s'avérer très difficile en raison d'un niveau élevé de dégénérescence : la base généralisée doit contenir toutes les "bonnes" variables pour rendre possible un pivot souhaitable. Nous observons souvent qu'un grand nombre de pivots dégénérés doivent être exécutés avant qu'un pivot non dégénéré puisse se produire. L'algorithme GISUD proposé dans cet article contourne ce problème en adoptant un point de vue géométrique.

3.2 Point de vue géométrique

L'espace des solutions GSPP $Conv(X)$ peut également être considéré comme un polyèdre dans $\mathbb{R}^n$. Une propriété bien connue de la programmation linéaire nous indique que si le GSPP est réalisable (c'est-à-dire $X \neq \emptyset$), alors un point extrême correspond à une solution optimale. Une propriété du SPP et du GSPP est que toute solution entière réalisable est un point extrême de X. Cela signifie que la solution optimale du GSPP est également un point extrême. Chaque point extrême de X correspond à une solution extrême et est associé à une ou plusieurs bases généralisées.

Deux points extrêmes (et donc deux solutions) x^1 et $x^2 \in X$ sont dits adjacents si et seulement si ils partagent une arête du polyèdre. Notons que l'adjacence des points extrêmes est différente de l'adjacence de la base généralisée. En effet, considérons $(B_{x^1}, L_{x^1}, U_{x^1})$ et $(B_{x^2}, L_{x^2}, U_{x^2})$, deux bases généralisées correspondant aux solutions x^1 et x^2, respectivement. Il se peut que x^1 et x^2 soient adjacents mais que $(B_{x^1}, L_{x^1}, U_{x^1})$ et $(B_{x^2}, L_{x^2}, U_{x^2})$ ne le soient pas (parce que les "mauvaises" variables ont été incluses dans B_{x^1}, et B_{x^2}). Cependant, si x^1 et x^2 sont adjacents, il existe deux bases généralisées $(\tilde{B}_{x^1}, \tilde{L}_{x^1}, \tilde{U}_{x^1})$ et $(\tilde{B}_{x^2}, \tilde{L}_{x^2}, \tilde{U}_{x^2})$, associées respectivement à x^1 et x^2, qui sont adjacentes. Dans ce cas, effectuer le pivot de $(\tilde{B}_{x^1}, \tilde{L}_{x^1}, \tilde{U}_{x^1})$ vers $(\tilde{B}_{x^2}, \tilde{L}_{x^2}, \tilde{U}_{x^2})$ équivaut à se déplacer de x^1 à x^2 le long de leur arête commune. La *direction* correspondant à ce déplacement est donnée par $d = x^2 - x^1$. Le *support* de d est l'ensemble des indices correspondant aux composantes non nulles de d, et est noté $Supp(d) = \{i \in [1, ..., n] | d_i \neq 0\}$.

Soit $X^I = \{x \in \{0,1\}^n | Ax = b\} \subseteq X$ l'ensemble des points extrêmes entiers (et donc des solutions entières). Balas et Padberg (1972) proposent de représenter le point extrême $x \in X^I$ par une *base de travail* $(\mathcal{P}_x, \mathcal{Z}_x)$, avec $\mathcal{P}_x = \{i \in \{1, .., n\} | x_i = 1\}$ et $\mathcal{Z}_x = \{i \in \{1, .., n\} | x_i = 0\}$. Contrairement à la base généralisée, cette représentation est unique pour tout point extrême de X^I. Notons qu'il suffit de spécifier $\mathcal{P}_x$ pour décrire de manière unique un point entier $x \in X^I$.

4 Algorithme GISUD

Une vue d'ensemble de l'algorithme GISUD est d'abord présentée, puis les différentes étapes de l'algorithme sont détaillées dans des sous-sections distinctes. L'algorithme GISUD est similaire à l'algorithme ISUD mais a été adapté pour résoudre le GSPP. La figure 1 présente un organigramme de l'algorithme. À partir d'une solution initiale, l'algorithme GISUD construit une séquence de solutions entières améliorées jusqu'à atteindre l'optimalité ou jusqu'à ce qu'un critère d'arrêt soit satisfait.

À chaque itération, GISUD tente d'améliorer la solution courante $x \in X^I$ en résolvant une séquence de problèmes auxiliaires de plus en plus coûteux en terme de temps de calcul. Chaque problème auxiliaire tente de trouver une solution améliorée, et l'itération termine lorsque l'un d'entre eux est résolu.

Tout d'abord, un *problème réduit* (RP) est résolu. Le RP tente de trouver un pivot améliorant non dégénéré d'une seule colonne de $\mathcal{Z}_x$ dans la représentation $(\mathcal{P}_x, \mathcal{Z}_x)$ associée à la solution actuelle. Si un tel pivot est trouvé, une solution améliorée est obtenue en effectuant ce pivot. La colonne correspondante entre dans la base généralisée et la structure $(\mathcal{P}_x, \mathcal{Z}_x)$ est mise à jour.

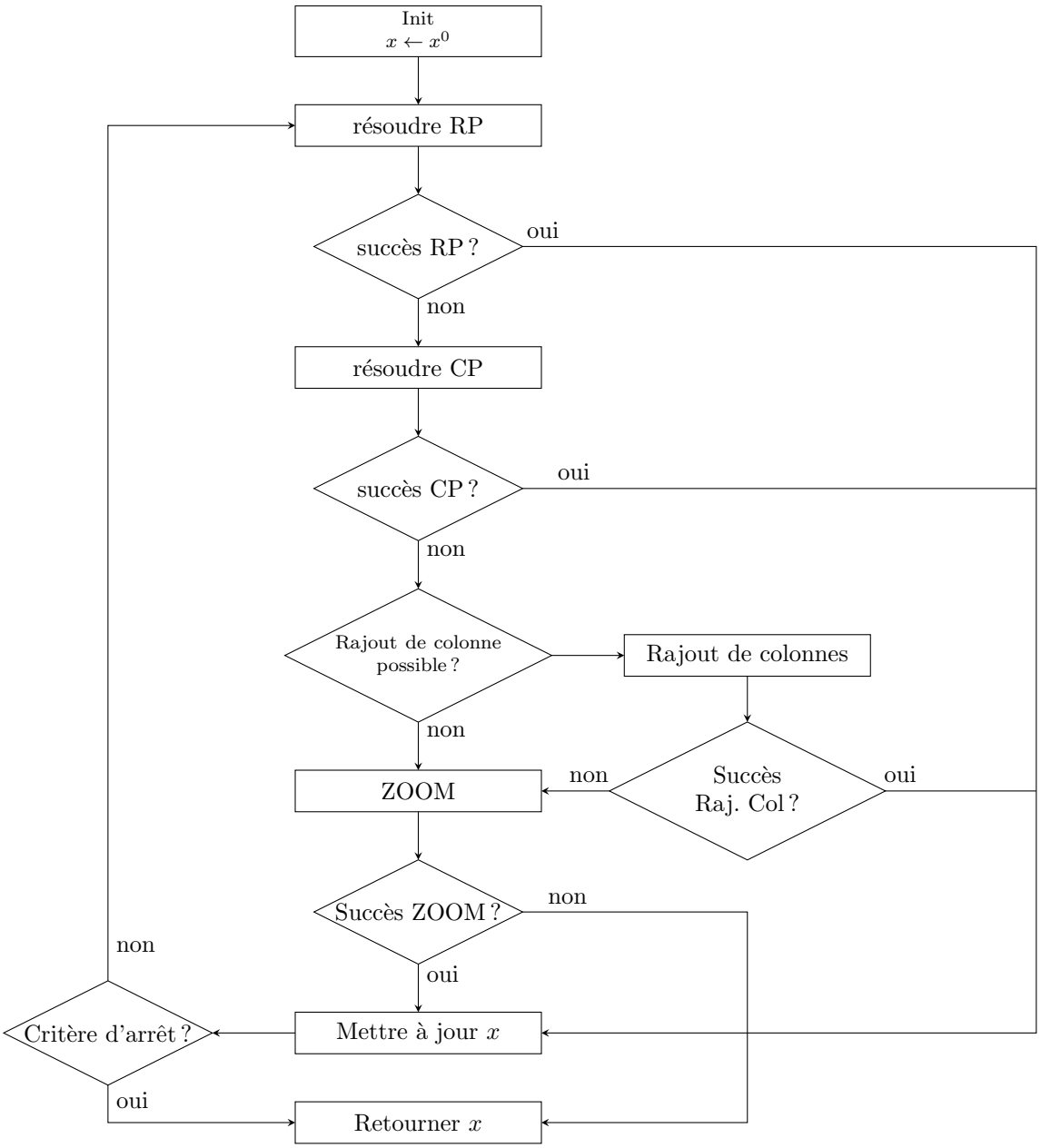

Figure 1 – Organigramme de l'algorithme GISUD

Lorsque le RP échoue, un *problème complémentaire* (CP) est résolu. Le CP recherche la meilleure direction d'amélioration parmi les arêtes du polyèdre adjacentes au point courant x. La direction renvoyée par le CP est souvent entière, auquel cas la solution courante (et la base de travail correspondante) est mise à jour et une nouvelle itération commence.

Si la solution de CP n'est pas entière, une *stratégie de rajout de colonnes* (présentée dans la section 4.3) peut être exécutée pour tenter d'obtenir une direction entière. En cas de succès, le point est mis à jour et une nouvelle itération débute. Il faut noter que la stratégie de rajout de colonnes ne peut être tentée que si la solution CP répond à un certain critère.

En dernier recours, une procédure *ZOOM* légèrement plus coûteuse (décrite dans la section 4.4) est appelée. Cette procédure renvoie une direction entière améliorante ou certifie que la solution courante est optimale. La solution est alors mise à jour en fonction de la direction renvoyée par cette procédure.

4.1 Problème réduit

Soit A_j la j-ème colonne de la matrice A. Soit C_x l'ensemble des indices de colonnes de $\mathcal{Z}_x$ qui sont *compatibles binaires* avec la solution courante x. La colonne A_j est compatible binaire avec x si et seulement si son pivot dans $\mathcal{P}_x$ mène à une solution entière $x^2 \in X^I$. Une définition formelle de la compatibilité binaire est présentée ci-dessous, ainsi qu'une proposition selon laquelle les pivots sur les colonnes compatibles binaires sont non dégénérés (ils conduisent à une solution entière différente de x).

Définition 2 (Compatibilité binaire)**.** La colonne A_k est dite compatible binaire avec $\mathcal{P}_x$ s'il existe un vecteur binaire $\alpha \in \{0,1\}^{|\mathcal{P}_x|}$ tel que $A_k = \sum_{j \in \mathcal{P}_x} \alpha_j A_j$ (c'est-à-dire que A_k est la somme de certaines colonnes de $\mathcal{P}_x$).

Proposition 1 (Pivot sur une colonne compatible binaire)**.** Considérons la solution $x \in X^I$ et la base de travail correspondante $\mathcal{P}_x$. Si une colonne A_k est compatible binaire avec $\mathcal{P}_x$, alors il existe un point extrême entier $x^2 \in X^I$ adjacent (au sens géométrique) à x pour lequel $k \in \mathcal{P}_{x^2}$.

Preuve. Comme A_k est compatible binaire, il existe $\tilde{\mathcal{P}} \subseteq \mathcal{P}_x$ tel que $A_k = \sum_{j \in \tilde{\mathcal{P}}} A_j$. Comme A_k a des composantes binaires, l'ensemble $\tilde{\mathcal{P}}$ est colonne-disjoint (les colonnes sont deux à deux orthogonales). Les colonnes associées forment alors un ensemble linéairement indépendant. On peut alors les compléter afin de former une base B_x et constituer une base généralisée (B_x, L_x, U_x) associée à x puisque x est un point extrême entier. Nous pouvons pivoter la colonne A_k dans B_x et ainsi obtenir un nouveau point extrême entier non nul x^2. Ce faisant, l'une des colonnes de $\tilde{\mathcal{P}}$ quitte B_x pour entrer dans U_x et les autres restent dans B_x mais prennent maintenant la valeur 0. x^2 est adjacent à x puisqu'un seul pivot permet de passer de x à x^2. □

Le RP détermine d'abord C_x, puis effectue un pivot sur une colonne compatible binaire avec un coût réduit négatif, s'il en existe. La colonne ayant le coût réduit le plus faible est choisie. Cela permet d'obtenir une solution entière améliorée, et la structure $(\mathcal{P}_x, \mathcal{Z}_x)$ est alors mise à jour.

En pratique, l'identification de toutes les colonnes compatibles binaires peut s'avérer coûteuse en termes de calculs. Pour accélérer l'algorithme, nous identifions un sous-ensemble de C_x en utilisant la méthode présentée dans la section 4.5. Notons que cela ne compromet pas l'optimalité de la méthode, car les autres problèmes auxiliaires (CP, rajout de colonnes, ZOOM) sont capables d'effectuer des pivots sur des colonnes compatibles. Soit x^2 la solution obtenue après une itération de RP réussie. Il est facile de montrer que $C_{x^2} \subseteq C_x$. Par conséquent, seules les colonnes précédemment compatibles binaires doivent être évaluées après une itération RP réussie.

4.2 Problème complémentaire

Le CP est résolu lorsque le RP échoue. Le CP recherche une direction améliorante d dans le cône des directions réalisables à partir de x, défini par $\Delta_x = \{d \in \mathbb{R}^n | Ad = 0, d_{\mathcal{P}_x} \leq 0, d_{\mathcal{Z}_x} \geq 0\}$, où $d_{\mathcal{P}_x}$ et $d_{\mathcal{Z}_x}$ sont les composantes de d correspondant à $\mathcal{P}_x$ et $\mathcal{Z}_x$, respectivement. Le CP est donné par (10)–(14).

$$CP(\mathcal{P}, \mathcal{Z}) = \min c_{\mathcal{P}} d_{\mathcal{P}} + c_{\mathcal{Z}} d_{\mathcal{Z}} \tag{10}$$

$$s.c$$

$$A_{\mathcal{P}} d_{\mathcal{P}} + A_{\mathcal{Z}} d_{\mathcal{Z}} = 0 \tag{11}$$

$$-w_{\mathcal{P}}^T d_{\mathcal{P}} + w_{\mathcal{Z}}^T d_{\mathcal{Z}} = 1 \tag{12}$$

$$d_{\mathcal{P}} \leq 0 \tag{13}$$

$$d_{\mathcal{Z}} \geq 0 \tag{14}$$

La fonction objectif (10) minimise le coût réduit de la direction. Les contraintes (11), (13) et (14) garantissent que la direction se trouve dans Δ_x. La contrainte (12) est une contrainte de normalisation qui permet de borner l'espace solution. Les poids $w_{\mathcal{P}}^T$ et $w_{\mathcal{Z}}^T$ peuvent prendre n'importe quelle valeur positive.

Le CP est résolu à l'aide de la méthode du simplexe. Pour cette raison, la solution optimale est nécessairement un point extrême du polyèdre (11)–(14).

Soit $Dom(CP)$ l'espace des solutions de CP. Les variables correspondent aux composantes d'une direction de Δ_x. La contrainte de normalisation (12) assure que le problème est borné mais permet aussi de restreindre la recherche aux arêtes. L'ensemble des directions correspondant aux déplacement le long d'arêtes est noté Δ_x^A. La preuve de ce point peut être faite grâce au théorème suivant.

Théorème 1 (Condition nécessaire et suffisante)**.** Soit $d \in \Delta_x \backslash \{0\}$ une direction reliant le point entier $x \in X^I$ au point extrême $y \in X$ (non nécessairement entier). La direction d correspond à une arête de $Conv(X)$ si et seulement si $\forall S \subset Supp(d),\ \{A_i, i \in S\}$ est linéairement indépendant.

Preuve.

Sens direct

Supposons que $d \in \Delta_x^A \backslash \{0\}$. Il existe donc $t_{max} > 0$ tel que $E = \{x + td, t \in [0, t_{max}]\}$ est une arête de $Conv(X)$, c'est-à-dire une 1-face.

Les faces d'un polytope peuvent être écrites comme l'intersection d'un sous-ensemble des contraintes qui sont saturées sur la face (en remplaçant les inégalités par des égalités). Sur l'arête E, les contraintes saturées sont :

— Toutes les contraintes d'égalité $Az = b$ (qui sont toujours saturées).

— Pour chaque $i \notin Supp(d)$, soit $x_i = 0$ soit $x_i = 1$ le long de l'arête, donc la contrainte correspondante ($x_i \geq 0$ ou $x_i \leq 1$, respectivement) est saturée.

La 1-face correspondant à l'arête est donc définie par :

$$E = \{\xi \in \mathbb{R}^n | A\xi = b,\ \xi_i = x_i \forall i \notin Supp(d),\ 0 \leq \xi \leq 1\}$$

La dimension affine de E est égale à celle de $\{x \in \mathbb{R}^{|Supp(d)|} | A_{Supp(d)} x = 0\}$. En effet, les autres composantes de x ne contribuent pas à la dimension car elles sont fixes. Puisque E est de dimension affine 1, on a que $Ker(A_{Supp(d)})$ est aussi de dimension 1. Par conséquent, $Rank(A_{Supp(d)}) = |Supp(d)| - 1$ (par le théorème du rang).

Cela signifie que nous pouvons extraire un sous-ensemble $J \subset A_{Supp(d)}$ tel que $|J| = |Supp(d)| - 1$ et J est linéairement indépendant. Comme $Ad = 0$, on a $A_{Supp(d)} d_{Supp(d)} = 0$. Par conséquent, nous pouvons échanger l'unique colonne de $A_{Supp(d)} \setminus J$ avec n'importe quelle colonne de J et nous aurons toujours un ensemble linéairement indépendant. Cela montre que tout sous-ensemble $J \subset A_{Supp(d)}$ est linéairement indépendant.

Sens réciproque

Soit $d \in \Delta_x \backslash \{0\}$. Supposons que pour tout $J \subset A_{Supp(d)}$, J est linéairement indépendant. Nous allons construire une base généralisée (B_x, L_x, U_x) associée au point x et faire pivoter une colonne dans cette base pour obtenir un nouveau point extrême $y = x + t_{max} d$. Soit $S \subset Supp(d)$ tel que $|S| = |Supp(d)| - 1$ et que $\{k\} = Supp(d) \setminus S$ désigne l'unique colonne de $Supp(d)$ non choisie dans S. La base généralisée associée à x est donnée par :

— $B_x = S \cup T$ with $T \subseteq [\![1, n]\!] \backslash S$ un ensemble d'indice de colonnes pour compléter la base

— $L_x = \{j \in [\![1, n]\!] \backslash B_x | x_j = 0\}$

— $U_x = \{j \in [\![1, n]\!] \backslash B_x | x_j = 1\}$

Rappelons que x est entier, donc toutes les variables sont soit à leur borne inférieure, soit à leur borne supérieure. Par indépendance linéaire de la base, B_x ne contient pas l'indice de colonne k. Cette colonne est alors soit dans L_x, soit dans U_x. On a alors $A_k = -\sum_{i \in S} \frac{d_i}{d_k} A_i$ car $Ad = 0$. Puisque la direction d est réalisable, on peut faire pivoter A_k à l'intérieur de B_x et obtenir un point $y = x + t_{max} d$ différent de x. Les points y et x sont donc adjacents. □

La preuve du sens réciproque du théorème 1 peut également être considérée comme un corollaire de la proposition 4 de IPS (Omer et al., 2015) dans le cas particulier où la base de travail utilisée dans IPS est vide et où toutes les colonnes sont incompatibles.

Le théorème 1 montre que la solution optimale de CP, notée d^*, correspond à une arête de $Conv(X)$ conduisant à un nouveau point extrême. En effet, d^* étant extrémale, on peut montrer l'indépendance linéaire des colonnes d'un sous-ensemble strict de son support. Le point extrême obtenu, en se déplaçant dans la direction d^*, peut être entier ou fractionnaire. S'il est entier, d^* est retourné et la solution courante est mise à jour. Il est facile de montrer que d^* est une direction entière si et seulement si ses composantes positives ont la même valeur. Dans la suite de cet article, nous notons $\Delta_x^I \subseteq \Delta_x$ l'ensemble des directions à partir de x qui sont entières.

4.3 Stratégie de rajout de colonnes

Dans cette section, nous montrons d'abord que le GSPP n'a pas la propriété de quasi-intégralité, comme c'est le cas pour le SPP. Nous présentons ensuite la stratégie de rajout de colonnes, qui permet, dans certains cas, à l'algorithme d'atteindre des points entiers qui ne sont pas directement adjacents au point courant x. Enfin, nous montrons que la stratégie de rajout de colonnes peut théoriquement permettre de recréer n'importe quelle arête de $Conv(X^I)$.

4.3.1 Quasi-intégralité

La propriété de quasi-intégralité stipule que toute arête de $Conv(X^I)$ est également une arête de $Conv(X)$. Elle implique qu'à partir de tout point sous-optimal x, il existe une direction entière améliorante qui est une arête de $Conv(X)$. Par conséquent, on peut trouver un meilleur point entier en restreignant la recherche aux voisins de x dans la relaxation linéaire. Le SPP possède la propriété de quasi-intégralité. Malheureusement, ce n'est pas le cas pour le GSPP : comme le montre l'exemple 1 ci-dessous : certains points entiers peuvent ne pas avoir de voisins entiers dans $Conv(X)$.

Exemple 1. Considérons l'instance GSPP définie par

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

Avec c quelconque.

On peut montrer que les deux seuls points entiers sont ;

$$x^1 = (1, 1, 1, 0, 0, 0, 0)^T$$

$$x^2 = (0, 0, 0, 1, 1, 1, 1)^T$$

Nous définissons également deux points fractionnaires :

$$x^3 = (1/2, 1/2, 1/2, 1, 1, 0, 0)^T$$

$$x^4 = (1/2, 1/2, 1/2, 0, 0, 1, 1)^T.$$

La direction $d^1 = (-1, -1, -1, 1, 1, 1, 1)^T$ joignant x^1 à x^2 n'est pas une arête du polytope car elle n'est pas extrême. En effet, $d^1 = d^2 + d^3$, avec

$$d^2 = (-1/2, -1/2, -1/2, 1, 1, 0, 0)^T \qquad d^3 = (-1/2, -1/2, -1/2, 0, 0, 1, 1)^T$$

De plus, d^2 et d^3 sont des directions réalisables car elles vont de x^1 à x^3 et x^4, respectivement.

4.3.2 Agrégation des colonnes de la solution de (CP)

Comme le GSPP n'a pas la propriété de quasi-intégralité, il peut y avoir des arêtes de $Conv(X^I)$ qui ne sont pas dans $Conv(X)$. Dans ce cas, la stratégie de rajout de colonnes peut étendre la recherche afin d'obtenir une direction entière qui ne se trouve pas directement dans Δ_x^A.

Une direction est appelée *direction entière minimale* partant de x si elle correspond à une arête de $Conv(X^I)$ adjacente à x. L'observation qui sous-tend la stratégie de rajout de colonnes est la suivante : Chaque direction entière minimale normalisée d partant de x peut être écrite comme une combinaison convexe de directions normalisées de Δ_x^A. De plus, le support de toute direction de cette combinaison convexe est inclus dans $Supp(d)$.

Lorsque le CP renvoie une direction fractionnaire d^*, la stratégie de rajout de colonnes suppose que $Supp(d^*)$ est inclus dans le support d'une direction entière, et tente de compléter ce support. Cependant, dans certains cas, nous pouvons montrer qu'aucune direction entière ne peut inclure $Supp(d^*)$. Dans ces cas la stratégie de rajout en colonne ne pourrait donc pas réussir. Plus précisément, la stratégie de rajout de colonnes ne peut être utilisée que si $\sum_{j \in Supp(d^*) \cap \mathcal{Z}_x} A_j \leq b$. C'est ce que montre la Proposition 2.

Proposition 2. Si d^I est une direction entière partant de x telle que $Supp(d^*) \subseteq Supp(d^I)$ alors $\sum_{j \in Supp(d^*) \cap \mathcal{Z}_x} A_j \leq b$.

Preuve de la proposition 2.

Soit x^2 le point entier obtenu en suivant d^I à partir de x. Nous savons que $Supp(d^I) \cap \mathcal{Z}_x$ est inclus dans $Supp(x^2)$ puisque ce sont les colonnes de $Supp(d^I)$ qui vont de 0 à 1. Puisque x^2 est entier, nous savons que $\sum_{i \in Supp(x^2)} A_i = b$. Nous avons que $Supp(d^*) \cap \mathcal{Z}_x \subseteq Supp(d^I) \cap \mathcal{Z}_x$, donc $Supp(d^*) \cap \mathcal{Z}_x \subseteq Supp(x^2)$. Nous concluons que $\sum_{j \in Supp(d^*) \cap \mathcal{Z}_x} A_j \leq b$. □

Lorsque d^* satisfait la condition de la proposition 2, la stratégie de rajout de colonnes est appliquée. En supposant que $Supp(d^*)$ est inclus dans le support d'une direction entière améliorante d^I, la stratégie de rajout de colonnes tente de trouver $Supp(d^I)$ au complet. Pour ce faire, les colonnes de $Supp(d^*)$ sont agrégées en une seule colonne artificielle qui s'ajoute à $GSPP_r$. Cette colonne artificielle est donnée par $C_x(I, J) = \sum_{i \in I} A_i - \sum_{j \in J} A_j$ avec $I = supp(d^*) \cap \mathcal{P}_x$ et $J = supp(d^*) \cap \mathcal{Z}_x$.

Le problème original $GSPP_r$ auquel est ajoutée la colonne $C_x(I, J)$ est appelé *problème augmenté*. Soit $n+1$ l'indice de $C_x(I, J)$ dans le problème augmenté. L'ajout de la colonne artificielle augmente de 1 la dimension affine de $Conv(X)$ et crée un nouveau point extrême le long de cette nouvelle dimension. En effet, le point x^A associé à $(\mathcal{P}_{x^A}, \mathcal{Z}_{x^A})$ avec

$$\mathcal{P}_{x^A} = (\{n+1\} \cup \mathcal{P}_x \setminus I) \cup J$$

$$\mathcal{Z}_{x^A} = (\mathcal{Z}_x \setminus J) \cup I$$

(i.e les colonnes de J entrent dans $\mathcal{P}_x$ avec la colonne $n+1$ et les colonnes de I sortent de $\mathcal{P}_x$). Il s'agit alors d'un point extrême du problème augmenté avec une composante non nulle le long de la nouvelle dimension. Le coût de $C_x(I,J)$ est $\sum_{j\in I} c_j - \sum_{j\in J} c_j$ donc le coût de x^A est égal à celui de x. Le poids w_{n+1} de $C_x(I,J)$ dans la contrainte de normalisation (12) du CP est $w_{n+1} = \sum_{j\in J} w_j + \sum_{j\in I} w_j$.

Un CP modifié, appelé MCP, est ensuite à nouveau résolu à partir du point artificiel x^A. La cohérence des coûts et des poids garantit qu'une direction qui améliore l'objectif de x^A améliore également l'objectif de x. Le MCP est limité aux directions correspondant aux arêtes adjacentes à x^A. Étant donné que l'objectif de MCP est de compléter le support connu : $I \cup J$, les colonnes de $I \cup J$ ne sont pas incluses dans MCP. MCP est formulé comme 15–21.

$$MCP(\mathcal{P}^A, \mathcal{Z}^A) = \min c_{\mathcal{P}^A} d_{\mathcal{P}^A} + c_{\mathcal{Z}^A} d_{\mathcal{Z}^A} \tag{15}$$

$$s.c$$

$$A_{\mathcal{P}^A} d_{\mathcal{P}^A} + A_{\mathcal{Z}^A} d_{\mathcal{Z}^A} = 0 \tag{16}$$

$$-w^T_{\mathcal{P}^A} d_{\mathcal{P}^A} + w^T_{\mathcal{Z}^A} d_{\mathcal{Z}^A} = 1 \tag{17}$$

$$d_{\mathcal{P}^A} \leq 0 \tag{18}$$

$$d_{\mathcal{Z}^A} \geq 0 \tag{19}$$

$$d_I = 0 \tag{20}$$

$$d_J = 0 \tag{21}$$

Soit $Dom(MCP)$ le domaine de MCP et d^{M*} sa solution. Notez que MCP a la même structure que CP. Toute direction $d \in Dom(MCP)$ a une direction correspondante $d^{eq} \in Dom(CP)$ avec le même coût réduit. En effet, si la colonne $n+1 \notin Supp(d)$, alors d est une solution réalisable de CP. En revanche, si $n+1 \in Supp(d)$, la direction correspondante d^{eq} est donnée par

$$d^{eq}_i = \begin{cases} d_{n+1} & \text{if } i \in I \cup J \\ d_i & \text{sinon} \end{cases}$$

On peut facilement vérifier que cette direction a le même coût réduit que $d \in Dom(MCP)$ et satisfait la contrainte de normalisation (12).

Si d^{M*} est une direction entière, elle peut contenir la colonne artificielle ou ne pas la contenir. Dans les deux cas, la direction CP correspondante est renvoyée et la solution actuelle est mise à jour. En revanche, si d^{M*} est fractionnaire, la stratégie de rajout de colonnes suppose que $Supp(d^{M*})$ donne des informations supplémentaires sur le support d'une direction entière. Soit $S = Supp(d^*) \cup Supp(d^{M*}) \setminus \{n+1\}$. Si S satisfait la condition de la proposition 2, une nouvelle colonne artificielle est construite à partir de S et MCP est à nouveau résolu. Sinon, la stratégie de rajout de colonnes échoue. La stratégie de rajout de colonnes est appelée de manière récursive jusqu'à ce qu'elle échoue ou renvoie une direction entière. Notez que $|S|$ augmente à chaque itération, ce qui garantit la terminaison de la stratégie de rajout de colonnes.

4.3.3 Existence de chemins

Dans cette section, nous montrons que toute direction entière minimale peut théoriquement être reconstruite par rajout successif de colonnes. En particulier, nous montrons qu'étant donné une direction entière minimale d^I reliant les points entiers x et y, il existe une séquence de directions normalisées $d^{1,norm}, d^{2,norm}, ..., d^{K,norm}$ qui sont des solutions de CP et MCP. La direction $d^{1,norm}$ est une solution du CP qui déclenche le rajout de colonnes. Si $K > 1$, les autres directions $d^{2,norm}, ..., d^{K,norm}$ sont des solutions du MCP résolues à chaque itération avec la dernière colonne rajoutée. La dernière direction $d^{K,norm}$ est entière et permet d'obtenir le point y.

Nous montrons d'abord dans la proposition 3 que pour toute direction entière d^I, il existe une direction $d \in Dom(CP)$ telle que $Supp(d) \subseteq d^I$.

Proposition 3. Soit d^I une direction entière normalisée partant de x de coût réduit négatif. Il existe au moins une direction extrême $d \in Dom(CP)$ telle que $Supp(d) \subseteq Supp(d^I)$ et $c^T d \leq c^T d^I$

Preuve de la proposition 3.

Sans perte de généralité, nous supposons que d^I, est normalisée avec les poids de CP w, de telle sorte que $d^I \in Dom(CP)$ (si ce n'est pas le cas nous normalisons simplement $d^I \leftarrow \frac{d^I}{w^T d^I}$). Comme d^I est une direction de $Dom(CP)$ elle peut s'écrire comme une combinaisons de points extrêmes $d^{ext,k}$ $k \in \mathcal{K}$ de CP.

$$d^I = \sum_{k \in \mathcal{K}} t_k d^{ext,k}$$

avec $t_k \in [0,1]$ et $\sum_{k \in \mathcal{K}} t_k = 1$. Comme les composantes de d^I et $d^{ext,k}$ pour $k \in \mathcal{K}$ sont positives sur $\mathcal{Z}_x$ et négatives sur $\mathcal{P}_x$, nous en déduisons que chaque direction de cette combinaison convexe $d^{ext,k}$ vérifie $Supp(d^{ext,k}) \subseteq Supp(d^I)$. □

On peut alors sélectionner une arête associée à une direction extrême $d^{1,norm} \in Dom(CP)$ dont le support est inclus dans $Supp(d^I)$. Si cette direction est proportionnelle à d^I, le chemin souhaité est obtenu. Sinon, $Supp(d^{1,norm}) \subset Supp(d^I)$ donc on ajoute une colonne $C_x(I,J)$ avec $I = Supp(d^{1,norm}) \cap \mathcal{P}_x$ et $J = Supp(d^{1,norm}) \cap \mathcal{Z}_x$ et on se déplace sur le nouveau point extrême artificiel créé. Soit x^A le point artificiel ajouté. Nous montrons ensuite (Proposition 4) que le point artificiel est "plus proche" de y que x (c'est-à-dire $|Supp(y - x^A)| < |Supp(y - x)|$).

Proposition 4. La direction $d^A = y - x^A$ vérifie $|Supp(d^A)| < |Supp(d^I)|$

Preuve de la proposition 4.

Nous supposons encore, sans perte de généralité, que d^I et d^A sont normalisés en fonction des poids de CP, (la normalisation n'affecte pas $Supp(d^I)$ et $Supp(d^A)$).

Soit $I = Supp(d^{1,norm}) \cap \mathcal{P}_x$ et $J = Supp(d^{1,norm}) \cap \mathcal{Z}_x$ Notons d'abord que $|I \cup J| \geq 2$ car le GSPP n'a pas de colonnes nulles. La direction d^A est une direction entière valide. Son support est $Supp(d^A) = (Supp(d^I) \backslash (I \cup J)) \cup \{n+1\}$. Nous avons alors que $|Supp(d^A)| < |Supp(d)|$ car $|I \cup J| \geq 2$. □

L'ajout de $C_x(I,J)$ préserve la structure du GSPP dans le problème augmenté. Les propositions 3 et 4 nous indiquent qu'à partir du point artificiel x^A, il est possible soit d'atteindre directement y, soit de trouver une autre direction $d^{2,norm}$ menant à un point artificiel strictement plus proche de y. De plus, si $c^T y < c^T x$ chaque direction de la séquence $d^{1,norm}, d^{2,norm}, ..., d^{K,norm}$ a un coût réduit négatif, ce qui signifie que toutes les arêtes associées ont un coût réduit négatif. Les résultats de cette sous-section peuvent être résumés avec le théorème 2.

Théorème 2. Soit d^I une direction minimale entière reliant x à y, avec $c^T y < c^T x$. Il est possible d'atteindre y à partir de x en résolvant un nombre fini de CP et MCP. De plus, la séquence de points artificiels visités est monotone en termes de coût (c'est-à-dire que chaque arête du chemin a un coût réduit négatif).

4.4 Procédure ZOOM

Si le CP et la stratégie de rajout de colonnes ne parviennent pas à trouver une direction entière, la procédure ZOOM est appelée. Semblable à la boucle principale de l'algorithme GISUD, la procédure ZOOM trouve une direction entière en résolvant alternativement un problème réduit (RP^{ZOOM}) et un

problème complémentaire (CP^{ZOOM}). Cependant, RP^{ZOOM} et CP^{ZOOM} diffèrent de RP et CP de plusieurs points. RP^{ZOOM} est un programme en nombres entiers (plutôt qu'un programme linéaire, comme RP), prend en entrée un ensemble de colonnes $\mathcal{P}_x^Z \supseteq \mathcal{P}_x$, et vise à trouver une direction entière parmi les colonnes qui sont *compatibles* avec cet ensemble $\mathcal{P}_x^Z$.

Définition 3. La colonne A_k est dite *compatible* avec l'ensemble $\mathcal{P}_x^Z$ s'il existe un vecteur de coefficients $\alpha \in \mathbb{R}^{|\mathcal{P}_x^Z|}$ tel que $A_k = \sum_{j \in \mathcal{P}_x^Z} \alpha_j A_j$.

La définition 3 est une relaxation de la compatibilité binaire (Définition 2), où les coefficients devaient être binaires. Cela permet de prendre en compte plus de colonnes dans RP^{ZOOM}, car il y a plus de colonnes compatibles que de colonnes compatibles binaires. On note $\mathcal{C}_{\mathcal{P}_x^Z}$ l'ensemble des colonnes compatibles et $\mathcal{I}_{\mathcal{P}_x^Z}$ l'ensemble des colonnes incompatibles.

RP^{ZOOM} est formulé comme il suit :

$$RP^{ZOOM}(\mathcal{P}_x^Z) = \min_z \; c^T_{\mathcal{C}_{\mathcal{P}_x^Z}} z \tag{22}$$

$$s.c.$$

$$A_{\mathcal{C}_{\mathcal{P}_x^Z}} z = b \tag{23}$$

$$z_j \in \{0,1\} \qquad \forall j \in \mathcal{C}_{\mathcal{P}_x^Z} \tag{24}$$

RP^{ZOOM} est nettement plus petit que le problème d'origine et peut être facilement résolu à l'aide d'algorithmes de programmation en nombres entiers traditionnels.

Au début de la procédure ZOOM, $\mathcal{P}_x^Z = \mathcal{P}_x$ (l'ensemble des variables positives dans la solution courante). Si RP^{ZOOM} trouve une direction d'amélioration, la solution courante est mise à jour et une nouvelle itération de GISUD débute. Sinon, il faut ajouter des variables incompatibles à RP^{ZOOM}. Pour ce faire, nous résolvons le problème complémentaire CP^{ZOOM} qui s'énonce comme il suit :

$$CP^{ZOOM}(\mathcal{P}_x, \mathcal{P}_x^Z) = \min_{d_{\mathcal{P}_x}, d_{\mathcal{P}_x^Z}} -c_{\mathcal{P}_x} d_{\mathcal{P}_x} + c^T_{\mathcal{C}_{\mathcal{P}_x^Z}} d_{\mathcal{I}_{\mathcal{P}_x^Z}} \tag{25}$$

$$s.c.$$

$$A_{\mathcal{P}_x} d_{\mathcal{P}_x} - A_{\mathcal{I}_{\mathcal{P}_x^Z}} d_{\mathcal{I}_{\mathcal{P}_x^Z}} = 0 \tag{26}$$

$$w^T_{\mathcal{I}_{\mathcal{P}_x^Z}} d_{\mathcal{I}_{\mathcal{P}_x^Z}} = 1 \tag{27}$$

$$d_{\mathcal{I}_{\mathcal{P}_x^Z}} \geq 0 \tag{28}$$

L'avantage de CP^{ZOOM} est qu'il implique uniquement des colonnes incompatibles. Il recherche une combinaison linéaire de variables incompatibles qui est compatible avec l'ensemble $\mathcal{P}_x$. On peut aussi montrer avec une preuve similaire à Zaghrouti et al. (2020) que si la solution de RP^{ZOOM} n'est pas optimale, alors CP^{ZOOM} a une solution de coût réduit strictement négatif. Soit d^* la solution de CP^{ZOOM} et $S = Supp(d^*) \cap \mathcal{I}_{\mathcal{P}_x^Z}$. Contrairement au cas du SPP, la direction d'amélioration représentée par d^* ne peut pas modifier le point courant (dégénérescence). Cependant, cela ne pose pas de problème puisque le but est uniquement de trouver des colonnes à rajouter au problème réduit. L'ensemble S est nécessairement non vide. Nous ajoutons S à $\mathcal{P}_x^Z$ et résolvons à nouveau RP^{ZOOM}. On itère entre RP^{ZOOM} et CP^{ZOOM} jusqu'à ce que l'on trouve un nouveau point entier différent de x ou jusqu'à ce que l'on ait prouvé l'optimalité par le fait que CP^{ZOOM} a un objectif positif ou nul.

En pratique, il est possible de réduire davantage la taille de RP^{ZOOM} en effectuant une réduction de contraintes. En effet, cela est possible puisqu'il n'y a que $Rank((A_j)_{j \in \mathcal{P}_x^Z})$ contraintes linéairement indépendantes. Cela contribue considérablement à accélérer la procédure ZOOM.

4.5 Approche multi-phase

Comme il a été fait dans l'implémentation de DCA (Elhallaoui et al., 2005) ou de ISUD (Zaghrouti et al., 2014), nous utilisons une approche multi-phases pour accélérer l'algorithme GISUD. Dans cette approche multi-phases, seules les colonnes "les moins incompatibles" avec la solution courante x sont considérées dans le problème complémentaire. Cela a pour effet de privilégier les directions entières. À mesure que la phase augmente, de plus en plus de colonnes incompatibles sont prises en compte. Le *degré d'incompatibilité* de chaque colonne A_k par rapport à la base de travail $\mathcal{P}_x$ est représenté par un entier positif ou nul $d_{k,\mathcal{P}_x}$.

Le CP est d'abord résolu en phase 1 : seules les colonnes ayant un degré d'incompatibilité inférieur ou égal à 1 sont considérées. Si l'objectif du CP est positif et que toutes les colonnes ne sont pas prises en compte dans le CP, la phase est augmentée et le CP est à nouveau résolu. Lorsque l'objectif CP devient strictement négatif, et que la direction est fractionnaire, soit la procédure de rajout de colonnes est déclenchée, soit la procédure de zoom est appelée (en gardant uniquement les colonnes de phase plus petite que la phase courante dans la procédure zoom). La phase est également augmentée lorsque la procédure de zoom échoue dans sa phase en cours. La phase est remise à 1 à chaque fois qu'une nouvelle itération démarre.

Cette approche multiphase est reliée à la mesure d'incompatibilité de chaque colonne $d_{k,\mathcal{P}_x}$. Intuitivement, $d_{k,\mathcal{P}_x}$ devrait être proportionnel à la taille de la plus petite déformation de $\mathcal{P}_x$ qui rend A_k compatible binaire. Dans une bonne mesure d'incompatibilité, les colonnes compatibles binaires devraient avoir une incompatibilité nulle. Plus $\mathcal{P}_x$ doit être "déformé" pour rendre la colonne compatible binaire, plus la mesure augmente.

L'ensemble des déformations considéré est la découpe des colonnes de $\mathcal{P}_x$ en plusieurs sous-colonnes. Remplacer une colonne de $\mathcal{P}_x$ par plusieurs sous-colonnes (en partitionnant les tâches couvertes) peut rendre A_k compatible binaire sans changer la compatibilité binaire des anciennes colonnes. Nous choisissons pour $d_{k,\mathcal{P}_x}$ le plus petit nombre de colonnes de $\mathcal{P}_x$ à diviser en deux afin d'obtenir la compatibilité binaire de la colonne A_k. Un ensemble de fractionnements de colonnes qui rendent A_k compatible est appelé *découpage compatible*.

Nous expliquons maintenant comment calculer l'incompatibilité de la colonne $A_k, k \in [\![1, n]\!] \backslash \mathcal{P}_x$. Soit W_j désignant l'ensemble des tâches couvertes par la colonne A_j. et soit $J_w = \{j \in \mathcal{P}_x | w \in W_j\}$ (le sous-ensemble de $\mathcal{P}_x$ correspondant aux colonnes qui couvrent w).

Pour trouver un découpage compatible, on assigne à chaque tâche couverte par la colonne A_k une colonne de $\mathcal{P}_x$. La colonne $j \in \mathcal{P}_x$ peut être affectée à la tâche w si et seulement si $j \in J_w$ (c'est-à-dire qu'elle couvre la tâche w). La définition 4 donne une définition formelle d'une affectation valide.

Définition 4 (Affectation valide)**.** Une affectation valide pour la colonne A_k et l'ensemble de colonnes $\mathcal{P}_x$ est une fonction f de W_k dans $\mathcal{P}_x$ telle que $f(w) \in J_w \forall w \in W_k$. L'image de f est notée $Im(f)$.

Notons que plusieurs affectations valides peuvent être possibles si au moins une des tâches couvertes par A_k a un membre droit supérieur à 1.

Étant donné une affectation f valide, un découpage compatible peut être obtenu en divisant chaque colonne $A_j \in \mathcal{P}_x$ en deux colonnes maximum : $A_j^1 = f^{-1}(\{j\})$ (où $f^{-1}(\{j\}) = \{w | f(w) = j\}$), et son complémentaire $A_j^2 = A_j \setminus A_j^1$. Nous définissons la taille d'un découpage par $s_{k,\mathcal{P}_x}(f) = \sum_{j \in Im(f)} \mathbb{1}(f^{-1}(\{j\}) \neq W_j)$ (le nombre de colonnes divisées).

Soit $\mathcal{F}_{k,\mathcal{P}_x}$ désignant l'ensemble des fonctions d'affectation valides pour l'ensemble de colonnes $\mathcal{P}_x$ et la colonne A_k. Le degré d'incompatibilité de la colonne A_k est :

$$d_{k,\mathcal{P}_x} = \min_{f \in \mathcal{F}_{k,\mathcal{P}_x}} s_{k,\mathcal{P}_x}(f)$$

Il correspond au nombre minimum de colonnes de $\mathcal{P}_x$ qui doivent être divisées pour rendre A_k compatible binaire.

Il est difficile de calculer $d_{k,\mathcal{P}_x}$ en pratique, car il s'agit d'un problème combinatoire. Au lieu de cela, nous approchons le degré d'incompatibilité avec l'algorithme suivant. On suppose qu'il existe un ordre total des tâches. Un tel ordre peut être obtenu, par exemple, en ordonnant les heures de début de chaque tâche. Soit $Succ(j, w)$ (resp. $Pred(j, w)$) la tâche qui suit (resp. précède) la tâche w dans la colonne j, selon l'ordre. Si aucune tâche ne suit (resp. ne précède) la tâche w dans la colonne j, on attribue la valeur NIL. La valeur du degré d'incompatibilité peut être approchée en résolvant le problème de plus court chemin suivant

Pour chaque tâche $w \in W_k$, nous créons des nœuds $|J_w|$, chaque nœud étant associé à une colonne de J_w et à la tâche w. Le nœud associé à la tâche w et à la colonne $j \in J_w$ est identifié par (w, j). Nous créons également un nœud source s et un nœud puit t. On connecte s à tous les nœuds correspondant à la première tâche de W_k. Ces arcs sont de longueur 0 pour les nœuds associés à une colonne commençant par la première tâche de W_k, et de longueur 1 pour les autres colonnes. On connecte également tous les nœuds correspondant à la dernière tâche de W_k à la destination par des arcs de longueur 0 si c'est la dernière tâche de la colonne, et 1 sinon. Pour deux tâches de W_k successives w_1, w_2, chaque nœud associé à w_1 est connecté à chaque nœud associé à w_2. La longueur de l'arc diffère selon la paire de nœuds : les nœuds (w_1, j_1) et (w_2, j_2) sont reliés par un arc de longueur L, qui est calculée par l'algorithme suivant. Si $Succ(j_1, w_1) \neq NIL$ et ($j_1 \neq j_2$ ou $Succ(j_1, w_1) \neq w_2$) on augmente L de 1. Si $Pred(j_2, w_2) \neq NIL$ et ($j_1 \neq j_2$ ou $Pred(j_2, w_2) \neq w_1$) on augmente L de 1. L vaut alors soit 0, soit 1, soit 2.

L'exemple suivant illustre la construction du graphe de plus court chemin. Considérons trois tâches, classées par $\{1, 2, 3\}$. La matrice de contraintes et les membres droits sont donnés par :

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

Supposons que la solution courante soit donnée par $\mathcal{P}_x = \{A_1, A_2, A_3\}$ (colonnes 1, 2 et 3 de A). Pour calculer $d_{4,\mathcal{P}_x}$, le degré d'incompatibilité de A_4 par rapport à $\mathcal{P}_x$, on résout le problème de plus court chemin illustré dans la figure 2.

Chaque chemin du graphe correspond à une affectation valide. En effet, les nœuds du graphe sont étagés : chaque niveau correspondant à une tâche $w \in W_k$. Chaque nœud d'un niveau est connecté aux nœuds du niveau suivant. Dans un chemin allant de s à t, exactement un nœud de chaque niveau apparaît. Le nœud sélectionné correspond à une colonne $j \in J_w$. Pour tout chemin donné, nous pouvons alors attribuer une colonne j de J_w à chaque tâche. Ceci définit une affectation valide et donc un découpage réalisable. Cependant, la longueur du chemin est supérieure ou égale à la taille du découpage. Par exemple, le chemin $(s; 1, 1; 2, 2; 3, 2; t)$ dans la figure 1 correspond à l'affectation $f(1) = 1, f(2) = 2, f(3) = 2$. La seule division de colonne dans cette affectation est la colonne 1 : les deux sous-colonnes sont $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ et $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ (les colonnes 2 et 3 ne sont pas divisées). Dans ce cas, la taille de coupe est la même que la longueur du chemin : 1.

Ce problème de plus court chemin est également utilisé pour identifier rapidement certaines colonnes compatibles binaires. Les colonnes qui ont un chemin optimal de longueur nulle dans leur graphe sont compatibles binaires puisque la longueur d'un chemin dans le graphe constitue une borne supérieure de la taille du découpage.

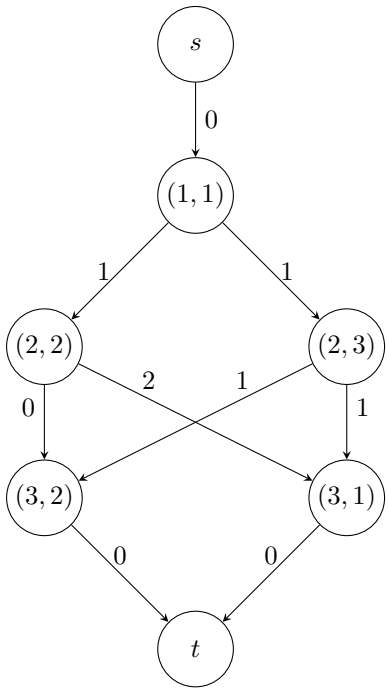

Figure 2 – Graphe orienté utilisé pour calculer le degré d'incompatibilité

5 Résultats

Cette section présente les résultats de l'algorithme GISUD sur plusieurs grandes instances de CPP. Nous présentons d'abord les détails de notre implémentation GISUD dans la section 5.1. La section 5.2 décrit les instances utilisées dans nos expériences. Les résultats sont ensuite présentés et analysés dans la section 5.3.

5.1 Détails d'implémentation

L'algorithme GISUD a été implémenté en C++. Les programmes linéaires et en nombres entiers impliqués dans l'algorithme GISUD (CP, MCP, RP^{ZOOM}, CP^{ZOOM}) sont résolus à l'aide de CPLEX 20.1. Les opérations d'algèbre linéaire ont été effectuées à l'aide de la bibliothèque C++ Eigen. Les poids de la contrainte de normalisation ont été choisis constants : $w_j = 1$ pour $j \in \mathcal{P} \cup \mathcal{Z}$.

Nous comparons GISUD avec le solveur CPLEX version 20.1 MIP avec les paramètres par défaut. Nous avons utilisé 8 threads pour CPLEX et GISUD. GISUD est une méthode séquentielle, mais l'utilisation de 8 threads accélère plusieurs de ses composants, à savoir la procédure ZOOM, le CP et le calcul du degré d'incompatibilité. Les algorithmes GISUD et CPLEX sont arrêtés lorsque l'écart entre le coût de la meilleure solution entière et la borne inférieure sur l'objectif est inférieur à 1%. Pour CPLEX, la borne inférieure est naturellement calculée et mise à jour tout au long de l'algorithme. Cependant, GISUD ne fournit pas de borne inférieure. Pour cette raison, nous calculons une borne inférieure au début de l'algorithme en résolvant la relaxation linéaire de GSPP à l'aide de CPLEX. Les résultats montrent que cette borne est très bonne. Cela est problement du au fait que nos instances contiennent un grand nombre de colonnes. Le temps passé à dériver cette borne inférieure est inclus dans les temps d'exécution globaux.

5.2 Instances

Nous avons testé GISUD sur plusieurs instances du CPP. Les plus grandes instances du CPP accessibles au public sont celles de Kasirzadeh et al. (2017). Il s'agit d'instances mensuelles destinées

aux pilotes d'une grande compagnie aérienne nord-américaine. Chaque instance correspond à un type d'avion différent. Malheureusement, puisque ces instances correspondent à des pilotes, les problèmes sont des SPPs (avec des seconds membres égaux à 1). Pour contourner cette limitation, nous créons 8 instances de GSPP dérivées des deux plus grandes instances de Kasirzadeh et al. (2017) (NW_320 et NW_757). Tout d'abord, deux semaines de données sont extraites de chaque instance pour créer au total quatre instances de CPP hebdomadaires. Ensuite, ces instances sont converties en GSPP en perturbant les seconds membres pour augmenter leurs valeurs. Deux perturbations ont été étudiées. La première (appelé RHS1) contient 33% de 1, 33% de 2 et 33% de 3. La seconde (appelée RHS2) contient 25% de 1, 25% de 2, 25% de 3 et 25% de 4. Ces différentes distributions permettent d'observer l'influence des seconds membres sur la difficulté des problèmes. RHS2 devrait être plus difficile puisque les seconds membres sont en moyenne plus élevés. Cela donne 8 instances qui sont répertoriées dans le tableau 1.

Les données originales répertorient les tâches mais n'incluent pas de colonnes. Nous générons des colonnes en résolvant chaque instance à l'aide de l'algorithme de génération de colonnes de Quesnel et al. (2017). Toutes les colonnes générées lors de l'exécution de l'algorithme ont été conservées.

Table 1 – Description des instances

Instance	Type de rhs	N cols	Nb. taches	Nb. zeros/col	Dist. Rhs			
					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

Pour chacune de ces instances, des solutions initiales de qualité différentes ont été générées. Ces solutions ont été générées en perturbant plus ou moins une solution presque optimale. Le processus de perturbation consiste à sélectionner aléatoirement deux colonnes actives (associées à des variables ayant des valeurs égales à 1) de la solution quasi optimale en les remplaçant par deux colonnes différentes couvrant les mêmes tâches globales. Le coût maximal des colonnes est alors donnée à chacune des nouvelles colonnes. Ce processus est répété jusqu'à ce qu'un certain pourcentage des colonnes initiales ait été remplacé. Nous avons considéré des solutions initiales avec 20%, 30% et 40% de leurs colonnes perturbées. Pour chaque instance, 4 solutions initiales ont été générées pour chaque taux de perturbation.

En pratique, de bonnes solutions initiales sont souvent disponibles. En effet, la génération de pairings est un processus hautement itératif dans lequel le planificateur résout une séquence de CPP, ajustant les paramètres et fixant et défixant manuellement des parties de la solution. Dans les cas où l'horaire des vols varie peu d'un mois à l'autre, la solution du mois qui précède peut également constituer une bonne solution initiale. Pour cette raison, de nombreuses colonnes de la solution optimale se retrouvent dans la solution initiale.

5.3 Résultats principaux

Nous avons exécuté GISUD et CPLEX sur toutes les instances. Les résultats sont présentés dans les tableaux 2, 3, 4 et 5, chaque tableau correspondant à une instance de base unique ainsi qu'un type de second-membres. Notez que chaque tableau présente les résultats pour les deux instances hebdomadaires dérivées de l'instance de base. Chaque ligne du tableau correspond à une instance. Les colonnes "Nom" et "Pert" correspondent au nom de l'instance et au taux de perturbation (en pourcentage). La colonne "Sol Initiale" correspond au coût de la solution initiale. Pour CPLEX, nous

rapportons la meilleure borne inférieure trouvée lors de la résolution (Meilleure borne), la valeur de la meilleure solution trouvée (Valeur Solution) ainsi que le temps d'exécution. Pour GISUD, nous rapportons la valeur de la meilleure solution trouvée avec le temps d'exécution, ainsi que des statistiques sur l'exécution de l'algorithme GISUD. Les colonnes "N its", "Z N" correspondent respectivement au nombre d'itérations de GISUD, au nombre d'appels de la procédure ZOOM. La colonne "AC N" indique le nombre de colonnes artificielles rajoutées, avec, entre parenthèses, le taux (en %) auquel la stratégie permet d'obtenir une direction entière. Les nombres entre parenthèses dans les colonnes "Valeur Solution" correspondent à l'écart en pourcentage par rapport à la meilleure borne inférieure de CPLEX.

Sans la stratégie de rajout de colonnes, GISUD améliore la solution courante en trouvant un voisin moins coûteux (distance de 1). Lorsque des colonnes sont rajoutées lors d'une itération, il est possible d'obtenir une meilleure solution plus éloignée du point courant. Les colonnes de la matrice associées aux variables qui changent de valeur entre le point courant et le nouveau point obtenu génèrent un espace vectoriel d'une certaine dimension D. En notant N le nombre de variables qui changent de valeur, la distance que nous avons utilisée pour caractériser cette distance est $N - D$: le nombre minimal de variables hors base qui doivent changer de valeur en choisissant une base généralisée au point courant qui contient les D colonnes linéairement indépendantes. La colonne "M.D" du tableau correspond à la distance maximale obtenue par GISUD lors de la résolution.

Table 2 – Résultats instances NW_320 avec RHS1

Nom	Pert.	Sol Initiale	CPLEX			GISUD					
			Meilleure borne	Valeur Solution	Temps (s)	Valeur Solution	Temps (s)	N its	Z N	AC N	M.D
320_1	20%	2164571	1001320	1003900 (0.26)	221	1011039 (0.96)	**46**	29	2	0 (-)	1
320_1	20%	2145016	1001310	1002860 (0.15)	133	1010310 (0.89)	**56**	34	4	0 (-)	1
320_1	20%	2155811	1001320	1002740 (0.14)	432	1011164 (0.97)	**64**	39	8	4 (25%)	2
320_1	20%	2136545	1001310	1003110 (0.18)	203	1010089 (0.87)	**80**	48	3	7 (57%)	6
320_2	20%	2509859	1048110	1049580 (0.14)	112	1058122 (0.95)	**69**	47	7	7 (28%)	3
320_2	20%	2556207	1048110	1049840 (0.16)	113	1058608 (0.99)	**55**	34	6	2 (100%)	4
320_2	20%	2507792	1048120	1049470 (0.13)	225	1058630 (0.99)	**75**	57	10	8 (12%)	4
320_2	20%	2541419	1048120	1049710 (0.15)	184	1058560 (0.99)	**60**	42	7	5 (40%)	2
320_1	30%	2738968	1001320	1003160 (0.18)	184	1011365 (0.99)	**97**	60	6	6 (33%)	2
320_1	30%	2778317	1001310	1002960 (0.16)	163	1011031 (0.96)	**95**	51	4	1 (100%)	2
320_1	30%	2662210	1001310	1003510 (0.22)	128	1011338 (0.99)	**112**	70	8	4 (25%)	2
320_1	30%	2737931	1001310	1002920 (0.16)	151	1011202 (0.98)	**134**	96	16	15 (40%)	3
320_2	30%	3287443	1048110	1049510 (0.13)	133	1057735 (0.91)	**130**	65	5	9 (55%)	2
320_2	30%	3116267	1048110	1049630 (0.14)	146	1058345 (0.97)	**121**	69	5	9 (55%)	5
320_2	30%	3304669	1048120	1049670 (0.15)	209	1058466 (0.98)	**125**	79	9	9 (55%)	5
320_2	30%	3255856	1048110	1049870 (0.17)	201	1058380 (0.97)	**138**	90	16	11 (27%)	3
320_1	40%	3451310	1001310	1002890 (0.16)	201	1011131 (0.97)	**155**	109	12	18 (66%)	7
320_1	40%	3370066	1001310	1003270 (0.20)	**144**	1010506 (0.91)	182	125	14	18 (83%)	6
320_1	40%	3365495	1001310	1002900 (0.16)	**131**	1009427 (0.80)	136	92	12	11 (63%)	4
320_1	40%	3421003	1001320	1002700 (0.14)	458	1010351 (0.89)	**151**	99	11	11 (63%)	4
320_2	40%	3979984	1048110	1049890 (0.17)	208	1058474 (0.98)	**198**	130	23	21 (57%)	5
320_2	40%	3963125	1048110	1050020 (0.18)	212	1052492 (0.42)	**195**	102	19	8 (37%)	4
320_2	40%	4061743	1048110	1049890 (0.17)	217	1054846 (0.64)	**185**	108	18	15 (53%)	8
320_2	40%	4046485	1048110	1049710 (0.15)	233	1058523 (0.98)	**153**	86	10	10 (60%)	3
		3010753	1024712	1026404 (0.16)	198	1034172 (0.91)	**117**	73.4	9.8	8.7 (4.6/8.7)	3

Table 3 – Résultats instances NW_320 avec RHS2

Nom	Pert.	Sol Initiale	CPLEX			GISUD					
			Meilleure borne	Valeur Solution	Temps (s)	Valeur Solution	Temps (s)	N its	Z N	AC N	M.D
320_3	20%	2745028	1257940	1258970 (0.08)	551	1270254 (0.97)	**109**	37	4	4 (50%)	5
320_3	20%	2790158	1257930	1259590 (0.13)	257	1269575 (0.92)	**124**	50	9	10 (70%)	4
320_3	20%	2644123	1257940	1261280 (0.26)	880	1269081 (0.88)	**131**	36	4	6 (83%)	3
320_3	20%	2632857	1257940	1261460 (0.28)	375	1269942 (0.95)	**111**	35	5	6 (83%)	4
320_4	20%	3292091	1374860	1379010 (0.30)	1405	1388647 (0.99)	**253**	64	12	14 (71%)	5
320_4	20%	3233742	1374860	1378430 (0.26)	1658	1388617 (0.99)	**130**	34	8	8 (50%)	7
320_4	20%	3277190	1374860	1377960 (0.22)	1546	1387573 (0.92)	**131**	31	7	4 (75%)	10
320_4	20%	3199304	1374860	1378730 (0.28)	1773	1385915 (0.80)	**173**	39	7	7 (28%)	4
320_3	30%	3499812	1257940	1259830 (0.15)	1084	1265670 (0.61)	**233**	81	9	14 (85%)	7
320_3	30%	3411146	1257950	1260480 (0.20)	1120	1270332 (0.97)	**217**	48	5	9 (55%)	3
320_3	30%	3401961	1257950	1261180 (0.26)	1022	1269693 (0.92)	**200**	63	3	13 (84%)	4
320_3	30%	3365347	1257940	1263540 (0.44)	1343	1270254 (0.97)	**152**	41	5	7 (71%)	3
320_4	30%	4203718	1374860	1380360 (0.40)	1849	1387931 (0.94)	**231**	67	14	13 (61%)	11
320_4	30%	4268044	1374860	1378480 (0.26)	1564	1387195 (0.89)	**230**	61	9	16 (56%)	6
320_4	30%	4171022	1374860	1378140 (0.24)	1631	1386530 (0.84)	**232**	72	12	14 (71%)	5
320_4	30%	4294081	1374860	1379110 (0.31)	2163	1381655 (0.49)	**261**	64	7	11 (90%)	4
320_1	40%	4258924	1257940	1261870 (0.31)	879	1264917 (0.55)	**338**	79	15	20 (55%)	9
320_1	40%	4370687	1257940	1261310 (0.27)	1282	1270548 (0.99)	**239**	73	14	18 (55%)	4
320_1	40%	4258692	1257940	1259340 (0.11)	801	1269535 (0.91)	**243**	84	10	8 (100%)	10
320_1	40%	4270003	1257950	1261050 (0.25)	1431	1270302 (0.97)	**366**	85	17	23 (52%)	7
320_4	40%	5255113	1374860	1378180 (0.24)	1622	1386672 (0.85)	**321**	76	16	19 (52%)	7
320_4	40%	5056570	1374860	1379050 (0.30)	1883	1384219 (0.68)	**341**	98	22	25 (64%)	4
320_4	40%	5250279	1374860	1380590 (0.42)	1659	1388539 (0.99)	**276**	72	13	15 (80%)	6
320_4	40%	5240708	1374860	1380660 (0.42)	2073	1380491 (0.41)	**285**	66	16	24 (41%)	9
		3849608	1316400	1319941 (0.27)	1327	1327670 (0.85)	**222**	60.7	10.1	12.8 (8.2/12.8)	5

Table 4 – Résultats instances NW_757 avec RHS1

Nom	Pert.	Sol Initiale	CPLEX			GISUD					
			Meilleure borne	Valeur Solution	Temps (s)	Valeur Solution	Temps (s)	N its	Z N	AC N	M.D
757_1	20%	1909699	907791	908839 (0.12)	257	916746 (0.98)	**51**	42	2	3 (100%)	3
757_1	20%	1947339	907795	909211 (0.16)	454	916949 (1.00)	**82**	74	9	12 (66%)	5
757_1	20%	1992396	907795	909443 (0.18)	397	916723 (0.97)	**65**	49	10	6 (16%)	2
757_1	20%	2008532	907804	909022 (0.13)	1234	916807 (0.98)	**78**	55	11	10 (20%)	3
757_2	20%	2159547	938597	940155 (0.17)	1010	946276 (0.81)	**56**	33	6	4 (75%)	2
757_2	20%	2065902	938581	939642 (0.11)	910	947312 (0.92)	**70**	37	11	5 (20%)	3
757_2	20%	2038708	938586	940495 (0.20)	1638	947216 (0.91)	**50**	29	2	5 (100%)	4
757_2	20%	2195301	938583	940918 (0.25)	2174	947684 (0.96)	**91**	38	6	6 (33%)	3
757_1	30%	2417050	907795	909475 (0.18)	435	915915 (0.89)	**90**	70	9	7 (71%)	5
757_1	30%	2485179	907790	910673 (0.32)	427	916643 (0.97)	**115**	88	12	12 (50%)	4
757_1	30%	2453481	907793	911454 (0.40)	198	916371 (0.94)	**104**	79	9	11 (72%)	4
757_1	30%	2453011	907800	909904 (0.23)	466	916846 (0.99)	**159**	88	12	10 (60%)	3
757_2	30%	2787612	938584	939989 (0.15)	1159	944030 (0.58)	**181**	60	10	8 (62%)	4
757_2	30%	2789456	938580	939764 (0.13)	915	947377 (0.93)	**101**	36	5	4 (100%)	6
757_2	30%	2667912	938579	939770 (0.13)	437	946163 (0.80)	**143**	57	9	10 (60%)	5
757_2	30%	2614080	938582	942268 (0.39)	2357	945946 (0.78)	**94**	49	6	6 (66%)	3
757_1	40%	2931914	907799	909659 (0.20)	432	916488 (0.95)	**145**	83	21	16 (31%)	3
757_1	40%	2964752	907796	909955 (0.24)	537	915774 (0.87)	**174**	104	22	16 (37%)	5
757_1	40%	2931712	907796	909194 (0.15)	231	916134 (0.91)	**108**	86	14	9 (44%)	3
757_1	40%	3022481	907796	908895 (0.12)	214	915464 (0.84)	**145**	96	15	11 (63%)	4
757_2	40%	3299400	938585	939719 (0.12)	367	947723 (0.96)	**184**	85	23	23 (43%)	3
757_2	40%	3354828	938596	940629 (0.22)	1706	945460 (0.73)	**179**	92	19	19 (63%)	5
757_2	40%	3351833	938584	940310 (0.18)	1009	946871 (0.88)	**169**	57	8	8 (62%)	7
757_2	40%	3339057	938586	939815 (0.13)	962	947325 (0.92)	**148**	77	20	9 (77%)	4
		2590882	923190	924966 (0.19)	830	931510 (0.89)	**116**	65.2	11.3	9.6 (5.2/9.6)	3

Table 5 – Résultats instances NW_757 avec RHS2

Nom	Pert.	Sol Initiale	CPLEX			GISUD					
			Meilleure borne	Valeur Solution	Temps (s)	Valeur Solution	Temps (s)	N its	Z N	AC N	M.D
757_3	20%	2530571	1178980	1181700 (0.23)	2498	1190376 (0.96)	**148**	58	8	7 (71%)	7
757_3	20%	2362958	1178970	1182080 (0.26)	2778	1190756 (0.99)	**150**	42	5	7 (42%)	4
757_3	20%	2414336	1178980	1183750 (0.40)	2499	1190657 (0.98)	**1121**	74	24	28 (17%)	5
757_3	20%	2543765	1178980	1181160 (0.18)	1998	1189839 (0.91)	**113**	53	5	7 (85%)	8
757_4	20%	2516674	1187680	1188780 (0.09)	765	1199064 (0.95)	**127**	50	8	12 (41%)	10
757_4	20%	2461914	1187690	1190340 (0.22)	3068	1197977 (0.86)	**134**	43	12	9 (22%)	2
757_4	20%	2512672	1187690	1191000 (0.28)	2987	1199211 (0.96)	**149**	36	4	7 (42%)	7
757_4	20%	2505038	1187680	1193620 (0.50)	725	1197551 (0.82)	**111**	38	9	5 (40%)	4
757_3	30%	3148295	1178970	1182250 (0.28)	2406	1186567 (0.64)	**186**	75	5	12 (91%)	4
757_3	30%	3102671	1178980	1185720 (0.57)	2574	1189495 (0.88)	**206**	80	12	20 (60%)	20
757_3	30%	3078211	1178980	1188820 (0.83)	2636	1185878 (0.58)	**199**	56	5	9 (77%)	3
757_3	30%	3053794	1178980	1182330 (0.28)	2492	1190604 (0.98)	**260**	80	17	15 (80%)	11
757_4	30%	3069951	1187690	1188760 (0.09)	1346	1198368 (0.89)	**119**	55	8	11 (72%)	6
757_4	30%	3113682	1187680	1189830 (0.18)	2175	1195698 (0.67)	**205**	67	13	15 (53%)	7
757_4	30%	3124066	1187680	1188750 (0.09)	654	1199504 (0.99)	**129**	62	8	20 (65%)	6
757_4	30%	3074170	1187690	1191060 (0.28)	3168	1199345 (0.97)	**128**	61	5	14 (64%)	4
757_3	40%	3776229	1178980	1182700 (0.31)	2398	1187317 (0.70)	**440**	123	21	32 (56%)	19
757_3	40%	3610644	1178970	1184010 (0.43)	1789	1189645 (0.90)	**270**	85	12	15 (73%)	6
757_3	40%	3658925	1178970	1183060 (0.35)	2731	1188027 (0.76)	**303**	124	17	22 (72%)	13
757_3	40%	3753432	1178970	1182170 (0.27)	2171	1188659 (0.82)	**228**	103	9	16 (100%)	13
757_4	40%	3844712	1187680	1189830 (0.18)	2529	1194319 (0.56)	**288**	98	19	27 (48%)	11
757_4	40%	3946385	1187690	1190240 (0.21)	2827	1198548 (0.91)	**217**	82	14	16 (68%)	22
757_4	40%	3801382	1187680	1189180 (0.13)	2351	1194847 (0.60)	**218**	65	10	13 (92%)	16
757_4	40%	3709618	1187690	1189210 (0.13)	1849	1198886 (0.93)	**189**	75	14	22 (59%)	5
		3113087	1183330	1186681 (0.28)	2225	1193380 (0.84)	**235**	70.2	11.0	15.0 (9.2/15.0)	8

Nous remarquons d'abord que GISUD est nettement plus rapide que CPLEX pour toutes les instances sauf deux. En fait, à l'exception des instances NW_320 avec RHS1, GISUD est en moyenne entre 5 et 10 fois plus rapide que CPLEX. Cependant, nous observons que les gaps d'intégrité atteints par CPLEX sont généralement inférieurs à ceux atteints par GISUD. En effet, CPLEX est une méthode duale qui atteint rarement l'intégralité. La différence entre deux solutions entières consécutives a tendance à être grande, de sorte que la première qui tombe en dessous du seuil de 1% peut présenter un petit écart d'optimalité. C'est également pourquoi nous observons une grande variance dans la valeur de la solution finale pour CPLEX. À l'inverse, GISUD est une méthode primale qui trouve une séquence de solutions entières de valeurs décroissantes. L'écart entre deux solutions entières consécutives est donc plus petit, ce qui explique que la première solution à tomber en dessous du seuil de 1% est souvent proche de ce seuil. Notez que nous observerions probablement le même comportement si le seuil était fixé à une valeur inférieure.

Nous observons que les temps d'exécution de GISUD sont plus faibles lorsque la solution initiale est de meilleure qualité (taux de perturbation plus faible). Cela montre que GISUD bénéficie grandement d'une bonne qualité de solution initiale. En revanche, les performances de CPLEX sont moins affectées par le taux de perturbation initial de la solution.

Le tableau 6 résume les résultats des tableaux 2–5 en présentant pour chaque instance de base et type de second-membre, les temps moyens de CPLEX et GISUD, le nombre de fois où la stratégie de rajout de colonnes a été appliquée, le taux de réussite de la stratégie de rajout de colonnes et la répartition des itérations entre CP, rajout de colonne et zoom.

Les temps de calcul sont plus faibles pour RHS1 que pour RHS2. Ceci était prévisible puisque RHS2 a des valeurs de second membres plus élevées, les instances correspondantes sont donc plus difficiles à résoudre. La supériorité de GISUD par rapport à CPLEX devient encore plus évidente à mesure que les valeurs des membres droits augmentent. Sur les instances RHS2, GISUD est plus de 5 fois plus rapide que CPLEX sur 320 RHS2 et plus de 9 fois plus rapide que cplex sur 757 RHS2. Sur les

instances RHS1, GISUD est 2 à 7 fois plus rapide que CPLEX. GISUD surpasse CPLEX en termes de temps de calcul sur presque toutes les instances.

Table 6 – Résumé des expérimentations

Nom	NW_320_RHS1	NW_320_RHS2	NW_757_RHS1	NW_757_RHS2
CPLEX temps moy.	198	1328	830	2225
GISUD temps moy.	117	222	116	235
N colonnes rajoutées	209	308	230	361
Succès rajout colonnes	53%	64%	54%	56%
Répartition des itérations CP/Raj. Col./ZOOM	1416/110/235	1016/197/243	1168/125/271	1200/221/264

Concernant la répartition des itérations, la majorité des itérations sont réalisées sans qu'il soit nécessaire d'appeler la procédure ZOOM ou la stratégie de rajout de colonnes : comme les instances ont des seconds membres proches de ceux du SPP, il existe de nombreuses arêtes conduisant à des solutions entières. En fait, à mesure que les seconds membres augmentent, la proportion d'itérations de ZOOM ou de rajout de colonnes augmente : la structure du problème s'éloigne de la propriété de quasi-intégralité. Sur les instances de type RHS2, le nombre d'itérations ZOOM est similaire au nombre d'itérations réussies de la stratégie de rajout de colonnes. Pour les instances de type RHS1, il y a environ deux fois moins d'itérations de rajout de colonnes réussies que d'itérations ZOOM. Cela montre que la condition requise pour déclencher le rajout de colonnes est souvent remplie en pratique, et que la stratégie de rajout de colonnes réussit à trouver une direction entière dans plus de 50% des cas lorsqu'elle est appelée.

5.4 Rajout de colonnes vs ZOOM

Afin de mesurer les bénéfices de la stratégie de rajout de colonnes, nous avons comparé la procédure de rajout de colonnes et la procédure ZOOM avec les mêmes conditions initiales. Pour ce faire, nous avons exécuté notre algorithme sur des instances NW_320 RHS2. Chaque fois que la stratégie de rajout de colonnes a réussi, nous sommes revenus en arrière et nous avons exécuté la procédure ZOOM à partir du même point de départ (la solution produite par la stratégie de rajout de colonnes est utilisée pour continuer l'algorithme). Nous avons ensuite calculé le *taux d'amélioration* des deux procédures. Le taux d'amélioration est défini comme la différence entre la nouvelle solution et la précédente, divisée par le temps passé dans la procédure (ZOOM ou rajout de colonnes).

Les résultats sont présentés dans le tableau 7. Chaque ligne correspond à une instance. Les colonnes "Nom" et "Pert" identifient chaque instance. La colonne "Nb. comp." indique le nombre de comparaisons effectuées pour chaque instance. Pour les procédures ZOOM et de rajout de colonnes, nous rapportons l'amélioration moyenne, la durée d'exécution moyenne de la procédure et le taux d'amélioration. Enfin, la colonne "Temps AC E" indique le temps moyen pris par la procédure de rajout de colonne lorsqu'elle échoue.

Nous observons que pour toutes les instances sauf une, le taux d'amélioration est meilleur pour la procédure de rajout de colonnes que pour ZOOM. De plus, la stratégie de rajout de colonnes a un taux d'amélioration deux fois plus important que ZOOM en moyenne.

Cela justifie l'intérêt de la stratégie de rajout de colonnes. Il est plus avantageux d'obtenir une direction d'amélioration en rajoutant des colonnes plutôt qu'en utilisant la procédure ZOOM. Ces résultats s'expliquent principalement par la structure primale de l'algorithme, adaptée à la ré-optimisation. La recherche locale d'une meilleure solution parmi les voisins de la solution courante est utilisée comme point clé de cet algorithme primal. Expérimentalement, nous observons que dans de nombreux cas, il est possible d'obtenir un tel voisin. De plus, le bon taux de réussite de la stratégie de rajout de colonnes observé permet d'éviter d'appeler la procédure ZOOM dans la plupart des cas.

Table 7 – Comparaison ZOOM et rajout de colonnes NW_320_RHS2

Nom	Pert.	Nb. comp.	ZOOM			Rajout col.			
			Amélioration	Temps (ms)	Taux Am.	Amélioration	Temps (ms)	Taux Am.	Temps AC E
320_3	20%	2	-7370	2869	-2568	-3978	361	**-7569**	164
320_3	20%	7	-3142	2795	-1123	-8279	324	**-14765**	551
320_3	20%	5	-598	2909	-205	-6696	316	**-17723**	307
320_3	20%	5	-2901	2835	-1023	-9832	530	**-17614**	141
320_4	20%	10	-746	3685	-202	-4630	381	**-923**	11587
320_4	20%	4	-535	3403	-157	-347	821	**-308**	302
320_4	20%	3	-15156	3556	-4262	-6972	428	**-10114**	783
320_4	20%	2	-1034	3616	**-285**	-2528	552	-149	6523
320_3	30%	12	-2514	2824	-890	-3763	433	**-7127**	569
320_3	30%	5	-846	2806	-301	-4027	461	**-423**	11315
320_3	30%	11	-2049	2888	-709	-2069	333	**-2306**	3098
320_3	30%	5	-14498	2777	-5220	-6725	306	**-17641**	187
320_4	30%	8	-7829	3535	-2214	-8825	390	**-6229**	1641
320_4	30%	9	-16823	3515	-4785	-24033	501	**-5782**	4698
320_4	30%	10	-7857	3534	-2223	-9597	565	**-12696**	476
320_4	30%	10	-4547	3571	-1272	-10347	272	**-31278**	583
320_3	40%	11	-649	2784	-233	-2841	479	**-581**	5385
320_3	40%	10	-24013	2753	-8721	-8728	447	**-10365**	493
320_3	40%	8	-809	2924	-276	-6506	459	**-14171**	0
320_3	40%	12	-2111	2819	-748	-2354	303	**-978**	2291
320_4	40%	10	-4076	3600	-1132	-12169	3971	**-2611**	765
320_4	40%	16	-7647	3506	-2180	-10491	1341	**-6173**	636
320_4	40%	12	-3031	3727	-812	-14950	380	**-26347**	747
320_4	40%	10	-15742	3617	-4351	-18809	381	**-4748**	2557
Average		8	-6167	3225	-1912	-8559	670	**-3830**	2777

6 Nombre minimal d'itérations de GISUD

Les travaux théoriques sur le SPP donnent une borne supérieure au nombre d'itérations d'ISUD nécessaires pour atteindre l'optimalité. Nous dérivons maintenant une borne similaire pour l'algorithme GISUD. Nous rappelons d'abord la borne de ISUD, qui est connectée à la conjecture de Hirsch, dans la section 6.1. Nous dérivons ensuite une limite pour le nombre d'itérations de GISUD dans la section 6.2.

6.1 Diamètre intégral du SPP et conjecture de Hirsch

Pour tout polytope donné, il est possible de construire un *graphe d'adjacence*, où chaque point extrême du polytope est représenté par un nœud, et deux nœuds sont connectés si et seulement s'il existe une arête contenant les deux points extrêmes. Un résultat classique de la programmation linéaire est la connexité de ce graphe : il est possible de passer d'un point extrême à un autre en se déplaçant uniquement le long des arêtes. Un tel chemin reliant x^1 et x^K avec comme points extrêmes intermédiaires $x^2, ..., x^{K-1}$ peut être représenté comme suit :

$$x^1 \xrightarrow{a_1} x^2 \xrightarrow{a_2} ... \xrightarrow{a_{K-1}} x^K$$

Dans la représentation sous forme de graphe, les arêtes reliant les points extrêmes sont notées $a_1, a_2, ..., a_{K-1}$. Un chemin est dit monotone si le coût diminue le long du chemin. La longueur minimale d'un chemin (nombre d'arêtes) reliant x^1 à x^K est appelée la *distance* entre x^1 et x^K. Le diamètre d'un polytope est alors défini comme la plus grande distance entre deux points extrêmes. Une forme de la conjecture de Warren M. Hirsch (Santos (2012)) est la suivante : dans un problème de programmation linéaire sous forme canonique (pour l'algorithme du simplexe) qui contient m^h contraintes, le diamètre du polytope correspondant est plus petit que m^h, c'est-à-dire que la distance entre deux points extrêmes est inférieure à m^h. Si elle est vraie, cette conjecture nous donne une borne supérieure sur le nombre minimum de pivots de simplexe nécessaires pour atteindre l'optimalité.

Considérons maintenant la restriction du graphe d'adjacence aux points extrêmes entiers seuls, appelé *graphe d'ajacence entier*. Egon Balas (Balas et Padberg (1972)) montre que le graphe d'ajacence entier est également connexe pour un polytope représentant la relaxation linéaire SPP. De plus, la conjecture de Hirsch est vérifiée dans ce cas.

6.2 Nombre minimal d'itérations de GISUD

Les résultats de la section 4.3.1 montrent que le graphe d'ajacence entier du GSPP n'est pas toujours connexe. Cependant, la section 4.3.3 montre que la connexité peut être restaurée en ajoutant des colonnes artificielles. On peut en fait généraliser la notion de chemin, étant donné qu'il s'agit uniquement de l'information d'une arête qui permet de rajouter un point artificiel et de se déplacer dessus. Par conséquent, il existe un chemin entre deux points entiers quelconques qui ne passe que par des points entiers réels ou artificiels. Nous pouvons pareillement définir la longueur de ce chemin comme le nombre d'arêtes utilisées : Les arêtes sont utilisées soit pour se déplacer directement le long de l'arête soit pour rajouter un point artificiel et se déplacer dessus. En notant x^1 comme point entier initial et $x^K = x^*$ comme point entier final, il existe une séquence d'arêtes $a_1, ..., a_{K-1}$ passant par un séquence de points entiers réels ou artificiels $x^1, x^2 ..., x^K$. Comme indiqué dans la section 4.3.3, déterminer quels points artificiels doivent être ajoutés nécessite la connaissance de x^K, ce qui n'est généralement pas le cas dans une application d'optimisation.

Pour calculer une limite sur le nombre d'itérations GISUD, nous considérons le polytope $Conv(X^{AUG})$, où X^{AUG} est l'ensemble des points extrêmes GSPP X augmenté des points artificiels générés tout au long de l'algorithme. Nous prouvons d'abord le lemme suivant.

Lemme 1. Soit $z \in X^I$ et x^* une solution optimale de GSPP . Il existe une direction entière minimale $d_z^{min} \in \Delta_z^I \backslash \{0\}$ telle que $Supp(d_z^{min}) \subseteq Supp(x^* - z)$ et $c^T d_z^{min} < 0$.

Preuve. Étant donné que x^* et z sont des points entiers, nous avons que $d^e = x^* - z \in \Delta_z^I \setminus \{0\}$ avec $c^T d^e < 0$. Si d^e est minimale entière, alors $d_z^{min} = d^e$. Sinon, d^e contient une sous-direction $d^{e,2} \in \Delta_z^I \backslash \{0\}$ telle que $Supp(d^{e,2}) \subset Supp(d^e)$ et $c^T d^{e,2} < 0$. $d^{e,2}$ est ensuite décomposé jusqu'à ce que nous obtenions une direction entière minimale. Cette direction est d_z^{min}. □

Proposition 5. Soit x^1, représenté par $(\mathcal{P}_{x^1}, \mathcal{Z}_{x^1})$, un point extrême entier et soit x^* un point extrême entier optimal. La distance entre x^1 et x^* dans la méthode GISUD (nombre minimal d'arêtes utilisées) est au plus $|Supp(x^* - x^1)| \leq 2 \sum_{i=1}^{m} b_i$.

Preuve. Soit $d^* = x^* - x^1$ la direction reliant x^1 à x^* et $S = Supp(d^*)$. Nous partitionnons S en $I = S \cap \mathcal{P}$ et $J = S \cap \mathcal{Z}$. Par le lemme 1, il existe une séquence de directions entières minimales d de x^1 à x^*. Cette séquence peut être représentée par $x^1 \xrightarrow{d_1} x^2 \xrightarrow{d_2} ... \xrightarrow{d_{K-1}} x^*$ où $\{x^i\} \setminus \{x^1\}$ désigne une séquence (éventuellement vide) de solutions entières intermédiaires. Notez que la direction d_k ne peut pas être associée à une arête de $Conv(X)$. Dans ce cas, le théorème 2 nous dit que $x^i \xrightarrow{d_i} x^{i+1}$ peut être remplacé par une séquence d'arêtes et de points extrêmes artificiels $\boldsymbol{x^i} \xrightarrow{a_1^1} x_1^1 \xrightarrow{a_2^1} x_2^1, ..., x_{n_i}^1 \xrightarrow{a_{n_i}^1} \boldsymbol{x^{i+1}}$. Ces arêtes sont des arêtes de $Conv(X^{AUG})$ formant un chemin dans la méthode de x^k à x^{k+1} et visitant des points extrêmes artificiels. On obtient alors le chemin monotone suivant :

$$\boldsymbol{x^1} \xrightarrow{a_2^1} x_2^1, ..., x_{n_i}^1 \xrightarrow{a_{n_1}^1} \boldsymbol{x^2} \xrightarrow{a_2^2} x_2^2, ..., x_{n_i}^2 \xrightarrow{a_{n_2}^2} \boldsymbol{x^{K-1}} \xrightarrow{a_2^{K-1}} x_2^{K-1}, ..., x_{n_i}^{K-1} \xrightarrow{a_{n_2}^{K-1}} \boldsymbol{x^K}$$

où d_k a été remplacé par une séquence d'arêtes a_1^k, ..., $a_{n_k}^k$ selon les cas. Les points non artificiels sont écrits en gras.

Pour $k \in [\![1, K-1]\!]$ le nombre d'arêtes entre x^k et x^{k+1} est inférieur à $|Supp(d^k)| - 1$. La longueur totale du chemin est inférieure ou égale à $\sum_{k=1}^{K-1} |Supp(d^k)| = |S|$ puisque $\{Supp(d^k) \forall k \in [\![1, K-1]\!]\}$

est une partition de S. La distance entre x^1 et x^* est inférieur ou égal à $|S| \leq |\mathcal{P}_x| + |\mathcal{P}_{x^*}|$. Puisque pour tout point entier, au plus $\sum_{i=1}^{m} b_i$ variables sont positives, nous concluons que le nombre d'arêtes utilisées est au plus $2 \sum_{je=1}^{m} b_i$. □

Notez également que le chemin créé dans la preuve de 5 est monotone : toutes les arêtes utilisées ont un coût réduit strictement négatif. Ceci est intéressant du point de vue de l'optimisation car cela montre que le chemin qui fournit la borne correspond à une séquence de solutions de valeurs décroissantes.

Pour comparer cette borne avec celle proposée par la conjecture de Hirsch, nous reformulons $GSPP_r$ sous la forme canonique du simplexe :

$$\min c^T x \tag{29}$$

$$s.c.$$

$$Ax = b \tag{30}$$

$$x_i + s_i = 1 \qquad \forall i \in \{0, 1, ..., n\} \tag{31}$$

$$x, s \geq 0 \tag{32}$$

où les contraintes 31 délimitent les variables avec s_i qui est une variable d'écart pour la contrainte de borne supérieure de la variable x_i. Cette formulation a $m^h = n + m$ contraintes et $2n$ variables. Soit $Dom(GSPP_r^{can})$ le polytope de la relaxation linéaire de formulation 29–32. Il est facile de montrer que tout chemin dans le polyèdre original a un chemin correspondant dans la reformulation canonique. Une base du GSPP contient au plus $n+m$ variables, donc seules $n+m$ variables peuvent changer leur valeur entre deux points entiers. On peut en conclure que la borne de la conjecture de Hirsch : $n+m$ est également satisfaite.

7 Conclusion

Dans cet article, nous abordons le GSPP, une classe de problèmes peu étudiée dans la littérature. Cette classe de problèmes est une généralisation du SPP. Nous proposons de résoudre le GSPP en utilisant un algorithme primal généralisant ISUD. L'algorithme GISUD améliore la solution courante en recherchant d'abord une meilleure solution entière parmi les solutions adjacentes dans la relaxation linéaire. Contrairement au SPP, le GSPP peut avoir des points entiers avec aucun voisins entiers. Nous avons donc proposé une nouvelle stratégie de rajout de colonnes qui permet de passer à des points entiers non directement adjacents au point courant. Nous montrons théoriquement qu'en utilisant cette stratégie de rajout de colonnes, il est toujours possible de trouver un chemin depuis n'importe quelle solution initiale vers la solution optimale. Nous montrons également qu'il existe un tel chemin dont la longueur ne dépasse pas $n + m$. Nous avons réalisé des expériences informatiques sur GISUD dans le cadre d'une réoptimisation de problèmes de pairings aérien. Les résultats montrent que l'algorithme GISUD est en moyenne plus de 5 fois plus rapide que CPLEX. Enfin, nous démontrons les avantages d'utiliser la stratégie de rajout de colonnes plutôt que la procédure ZOOM.

Références

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

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price : Column generation for solving huge integer programs. Operations Research 46(3) :316–329, http://dx.doi.org/10.1287/opre.46.3.316.

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

Elhallaoui I, Villeneuve D, Soumis F, Desaulniers G (2005) Dynamic aggregation of set-partitioning constraints in column generation. Operations Research 53(4) :632–645, http://dx.doi.org/10.1287/opre.1050.0222.

Feo TA, Resende MG, Smith SH (1994) A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42(5) :790–986, https://doi.org/10.1287/opre.42.5.860.

Firla R, Haus UU, Spille B, Weismantel R (2001) Integer pivoting revisited .

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

Haus U, Köppe M, Weismantel R (2001) A Primal All Integer Algorithm Based on Irreducible Solutions. Preprint (Univ., Fak. für Mathematik), https://books.google.ca/books?id=_Q75vgEACAAJ.

Holland JH (1992) Genetic algorithms computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. Scientific American 66–72.

Hussain K, Mohd Salleh MN, Cheng S, Shi Y (2019) Metaheuristic research : a comprehensive survey. Artificial intelligence review 52 :2191–2233.

Kasirzadeh A, Saddoune M, Soumis F (2017) Airline crew scheduling : Models, algorithms, and data sets. EURO Journal on Transportation and Logistics 6(2) :111–137, http://dx.doi.org/10.1007/s13676-015-0080-x.

Letchford A, Lodi A (2002) Primal cutting plane algorithms revisited. Mathematical Methods of Operational Research 56 :67–81, http://dx.doi.org/10.1007/s001860200200.

Omer J, Rosat S, Raymond V, Soumis F (2015) Improved primal simplex : A more general theoretical framework and an extended experimental analysis. INFORMS Journal on Computing 27(4) :773–787, http://dx.doi.org/10.1287/ijoc.2015.0656.

Pan PQ (1998) A basis-deficiency-allowing variation of the simplex method for linear programming. Computers & Mathematics with Applications 36(3) :33–53, http://dx.doi.org/https://doi.org/10.1016/S0898-1221(98)00127-8.

Quesnel F, Desaulniers G, Soumis F (2017) A new heuristic branching scheme for the crew pairing problem with base constraints. Computers & Operations Research 80 :159–172.

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

Rosat S, Elhallaoui I, Soumis F, Chakour D (2017) Influence of the normalization constraint on the integral simplex using decomposition. Discrete Applied Mathematics 217 :53–70, http://dx.doi.org/https://doi.org/10.1016/j.dam.2015.12.015, combinatorial Optimization : Theory, Computation, and Applications.

Santos F (2012) A counterexample to the hirsch conjecture. Annals of Mathematics 176(1) :383–412, http://www.jstor.org/stable/23234170.

Spille B, Weismantel R (2005) Primal integer programming. Aardal K, Nemhauser G, Weismantel R, eds., Discrete Optimization, volume 12 of Handbooks in Operations Research and Management Science, 245–276 (Elsevier), http://dx.doi.org/https://doi.org/10.1016/S0927-0507(05)12005-2.

Stützle T (1998) Local search algorithms for combinatorial problems. Darmstadt University of Technology PhD Thesis 20.

Tahir A, Desaulniers G, El Hallaoui I (2019) Integral column generation for the set partitioning problem. EURO Journal on Transportation and Logistics 8(5) :713–744, http://dx.doi.org/https://doi.org/10.1007/s13676-019-00145-6.

Tahir A, Desaulniers G, El Hallaoui I (2022) Integral column generation for set partitioning problems with side constraints. INFORMS Journal on Computing 34(4) :2313–2331, http://dx.doi.org/10.1287/ijoc.2022.1174.

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

Trubin V (1969) On a method of solution of integer linear programming problems of a special kind. Soviet Mathematics Doklady, volume 10, 1544–1546.

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

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

Zaghrouti A, El Hallaoui I, Soumis F (2018) Improved integral simplex using decomposition for the set partitioning problem. EURO Journal on Computational Optimization 6(2) :185–206, http://dx.doi.org/https://doi.org/10.1007/s13675-018-0098-6.

Zaghrouti A, Elhallaoui I, Soumis F (2020) Improving set partitioning problem solutions by zooming around an improving direction. Annals of Operations Research 284, http://dx.doi.org/10.1007/s10479-018-2868-1.

Zaghrouti A, Soumis F, Hallaoui IE (2014) Integral simplex using decomposition for the set partitioning problem. Operations Research 62(2) :435–449, http://www.jstor.org/stable/24540586.