

Titre: Génération de colonnes en nombres entiers pour les problèmes de
Title: type partitionnement d'ensemble

Auteur: Adil Tahir
Author:

Date: 2019

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Tahir, A. (2019). Génération de colonnes en nombres entiers pour les problèmes
Citation: de type partitionnement d'ensemble [Thèse de doctorat, Polytechnique Montréal].
PolyPublie. <https://publications.polymtl.ca/4102/>

 Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/4102/>
PolyPublie URL:

**Directeurs de
recherche:** Issmaïl El Hallaoui, & Guy Desaulniers
Advisors:

Programme: Doctorat en mathématiques
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Génération de colonnes en nombres entiers pour les problèmes de type
partitionnement d'ensemble**

ADIL TAHIR

Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Décembre 2019

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Génération de colonnes en nombres entiers pour les problèmes de type
partitionnement d'ensemble**

présentée par **Adil TAHIR**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Louis-Martin ROUSSEAU, président

Issmaïl EL HALLAOUI, membre et directeur de recherche

Guy DESAULNIERS, membre et codirecteur de recherche

François SOUMIS, membre

Elina RÖNNBERG, membre externe

DÉDICACE

À ma mère Naima Anssari, ma source d'inspiration, l'amour de ma vie. Tu m'as comblé avec ta tendresse et affection tout au long de mon parcours. En ce jour mémorable, pour moi ainsi que pour toi, reçoit ce travail en signe de ma vive reconnaissance et ma profonde estime. Puisse le bon Dieu te donner santé, bonheur et longue vie afin de te combler à mon tour.

...

À mon père Mustapha Tahir, aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que tu as consentis pour mon instruction. Que ce modeste travail soit l'exaucement de tes vœux tant formulés, le fruit de tes innombrables sacrifices, bien que je ne t'en acquitterai jamais assez.

...

À mon épouse Omaïma Shayah, aucun mot ne pourrait exprimer ma gratitude, mon amour et mon respect. Je remercie le bon Dieu qui a fait croiser nos chemins.

...

À mes frères Ezzitouni, Sabir, Mouaad et Saad, je suis fier de vous, je vous aime tous.

...

REMERCIEMENTS

Mes remerciements les plus profonds vont à M. Issmaïl El Hallaoui et M. Guy Desaulniers mes directeurs de recherche pour leur encadrement exceptionnel, leur confiance ainsi que leur support continu et inconditionnel. Je les remercie de m'avoir appris l'organisation du travail, la discipline et la rigueur. Je les remercie également pour m'avoir accueilli au GERAD et m'avoir donné l'opportunité d'accomplir cette thèse sous une supervision de haute qualité.

Je tiens à exprimer ma gratitude à Mohammed Saddoune, mon professeur à la Faculté des Sciences et Techniques de Mohammedia au Maroc. Je lui serai toujours reconnaissant de m'avoir permis de franchir le premier pas vers l'intégration au sein d'une agréable équipe de recherche.

Je remercie chaleureusement mes beaux-parents Driss Shayah et Aicha Dahiri pour leur encouragement, leur support ainsi que leur confiance.

J'adresse mes sincères remerciements à Rachid Hassani, Abdelouahab Zaghrouti, Ilyas Himich et Mayssoun Messaoudi pour leur aide et leurs conseils durant les moments difficiles. Aussi, je n'oublierai jamais les autres amis du GERAD avec qui j'ai établi une relation de fraternité et de collaboration. Ainsi, je remercie Mouad Mourabit, Tayeb Mhamedi, Safae Er-Rbib, Abderrahman Bani, Issoufou Abdou Amadou, Salah-eddine Makhloufi et Yassine Yaakoubi.

Je tiens à exprimer ma gratitude à tous ceux et celles qui ont contribué de près ou de loin à l'accomplissement de cette thèse.

RÉSUMÉ

Le problème de partitionnement d'ensemble (SPP : Set Partitioning Problem) est un problème d'optimisation combinatoire, se formulant comme un modèle de programmation linéaire en nombres entiers. L'objectif du SPP est de trouver une partition d'un ensemble de tâches (e.g., vols, clients, ...) à coût minimum. Le modèle du SPP est composé de deux types de contraintes : les contraintes de partitionnement qui assurent que chaque tâche soit couverte exactement une fois, et des contraintes supplémentaires qui permettent de modéliser d'autres aspects du problème traité comme les disponibilités du personnel. Le SPP permet de modéliser plusieurs problèmes rencontrés dans l'industrie comme les problèmes de tournées de véhicules et de rotations d'équipages dans le domaine aérien (CPP : crew pairing problem).

Les problèmes mentionnés ci-dessus sont généralement résolus à l'aide de la méthode *Branch-and-Price*. C'est une méthode duale fractionnaire dont l'objectif est de trouver des solutions entières en ignorant, dans un premier temps, les contraintes d'intégralité sur les variables du modèle de SPP. Pour ce faire, *Branch-and-Price* explore un arbre de branchement où chaque nœud de branchement est résolu à l'aide de la méthode de génération de colonnes. Malgré sa popularité, *Branch-and-Price* peut être lente dans le cas de problèmes difficiles de très grande taille. Par conséquent, les solutions entières recherchées ne sont obtenues que vers la fin de la résolution.

D'autres méthodes, dites primales, sont également utilisées pour résoudre le SPP. Elles cherchent une solution optimale en améliorant à chaque itération la solution entière courante. À l'itération 0, ces méthodes partent d'une solution entière initiale qui est généralement disponible pour les problèmes industriels. Parmi les méthodes primales efficaces, on trouve l'algorithme du *simplexe en nombres entiers avec décomposition* (ISUD) et ses deux variantes améliorées nommées ZOOM et I²SUD. Cet algorithme primal trouve une séquence de solutions entières avec des coûts décroissants, menant à une solution optimale ou quasi-optimale. ISUD est l'un des algorithmes primaux potentiels qui a pu résoudre efficacement des problèmes SPP de grande taille. Ceci a vivifié l'intérêt que l'on commence à porter aux approches primales. Malgré ses deux variantes améliorées, ISUD a cependant deux limitations majeures : i) il suppose que les colonnes sont générées a priori, ii) il ne gère pas les contraintes supplémentaires, souvent rencontrées en milieux pratiques. Ceci dit, et à l'instar de ces constats, nous proposons dans le cadre de cette thèse trois sujets pour remédier aux limitations précédemment soulevées.

Dans le premier sujet, nous proposons une méthode de génération de colonnes en nombres

entiers (ICG) qui combine ISUD et la génération de colonnes pour résoudre des problèmes de partitionnement avec un très grand nombre de variables. ICG génère une séquence de solutions entières avec des coûts décroissants, menant à une solution optimale ou quasi-optimale. Des expérimentations numériques sur des instances du CPP et du problème d’horaires de chauffeurs et d’autobus (VCSP : vehicle and crew scheduling problem), impliquant jusqu’à 2000 contraintes montrent que ICG surpasse nettement deux heuristiques populaires de génération de colonnes. ICG trouve des solutions optimales ou quasi-optimales en moins d’une heure de temps de résolution, en générant jusqu’à 300 solutions entières pendant le processus de résolution. Cependant, ICG ne permet pas de résoudre les SPPs avec des contraintes supplémentaires, d’où le deuxième sujet de cette thèse.

Dans le deuxième sujet, nous proposons une version améliorée d’ICG, notée I²CG, qui peut résoudre efficacement des problèmes de partitionnement d’ensembles de grande taille avec des contraintes supplémentaires. Ces dernières sont difficiles à gérer dans le contexte des méthodes primales car elles peuvent créer des minimums locaux en coupant tous les points extrêmes entiers adjacents à la solution courante. Les expérimentations numériques sur des instances du CPP impliquant jusqu’à 1761 contraintes montrent que I²CG est efficace et surpasse largement deux heuristiques populaires de génération de colonnes. Pour la plus grande instance testée, I²CG a pu produire en moins d’une heure de temps de calcul plus que 500 solutions entières menant à une solution optimale ou quasi-optimale.

Le troisième sujet porte sur l’amélioration de la méthode I²CG à l’aide des techniques d’apprentissage machine. En effet, nous utilisons un modèle de prédiction pour prédire avec une certaine probabilité les connexions entre les vols dans le CPP. Ces probabilités sont incorporées dans la méthode I²CG pour favoriser la génération des colonnes ayant une plus grande probabilité d’appartenir à des solutions optimales ou quasi-optimales. Les tests effectués sur 49 instances de CPP, impliquant jusqu’à 1740 vols, montrent clairement que la nouvelle version d’I²CG est plus rapide que I²CG. En effet, la nouvelle version génère et trouve aussi une séquence de solutions entières avec des coûts décroissants jusqu’à atteindre une solution optimale ou quasi-optimale en réduisant le temps de résolution total de 50% en moyenne.

ABSTRACT

The Set Partitioning Problem (SPP) is a combinatorial optimization problem, formulated as an integer linear programming model. The SPP aims to find a minimum cost partition of a set of tasks (e.g., flights, customers, ...). The SPP model has two types of constraints: partitioning constraints ensuring that each task is covered exactly once; and side constraints for modeling other aspects of the problem, such as staff availability. The SPP can be used to model several real-world problems such as vehicle routing and crew pairing problems (CPP).

The problems mentioned above are usually solved using *Branch-and-Price* based approaches. *Branch-and-Price* is a fractional dual method that aims to find integer solutions by ignoring the integrality constraints in the SPP model. *Branch-and-Price* explores a search tree where each branch node is solved using the column generation method (CG). Despite its wide popularity, *Branch-and-Price* becomes increasingly slow on very large and difficult problems. Therefore, the expected integer solutions are only obtained in the end of the resolution.

Other methods, known as primal methods, are also used to solve the SPP. They look for optimality by improving the current integer solution. These methods start from an initial integer solution which is generally available in industrial contexts. One of the most efficient primal methods is the *integral simplex using decomposition* (ISUD) algorithm and its two improved variants called ZOOM and I²SUD. This primal algorithm finds a sequence of integer solutions with decreasing costs, leading to an optimal or near-optimal solution. ISUD is one of the potential primal algorithms that has been able to efficiently solve large SPPs, and so giving a boost of interest towards primal approaches. Even with its two improved variants, ISUD still has two main limitations: i) it assumes that the columns are generated a priori, ii) it does not handle the side constraints commonly encountered in practice.

In the first subject of this thesis, we develop an integral column generation (ICG) heuristic that combines ISUD and column generation to solve SPPs with a very large number of variables. Computational experiments on instances of the public transit vehicle and crew scheduling problem (VCSP) and of CPP involving up to 2000 constraints show that ICG clearly outperforms two popular column generation heuristics (RMH : restricted master heuristic and DH : diving heuristic). ICG can yield optimal or near-optimal solutions in less than one hour of computational time, generating up to 300 integer solutions during the solution process. However, ICG cannot solve SPPs with side constraints.

In the second subject, we develop a generalized version of ICG, denoted I²CG, that can solve efficiently large-scale set partitioning problems with side constraints (SPPSC). The

latter are difficult to handle in the context of primal methods because it can create local minima by cutting all integer extreme points adjacent to the current solution. Computational experiments on instances of the airline crew pairing problem involving up to 1761 constraints show that I²CG is efficient and significantly outperforms DH and RMH. For the largest tested instance, I²CG can produce more than 500 integer solutions leading to an optimal or quasi-optimal (i.e., near-optimal) solution.

The third subject proposes a machine learning based improvement process for the I²CG algorithm. Indeed, we use a prediction model to predict with some probability the connections between flights. These probabilities are embedded in the I²CG algorithm to favour the generation of columns with a higher probability of being in the optimal or near-optimal solutions. Computational experiments on 49 instances of the CPP involving up to 1740 flights show that the new version of I²CG is faster than I²CG. It generates a sequence of integer solutions with decreasing costs until reaching an optimal or quasi-optimal solution and the computational time is reduced by 50% on average.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
TABLE DES MATIÈRES	ix
LISTE DES TABLEAUX	xii
LISTE DES FIGURES	xiii
LISTE DES SIGLES ET ABRÉVIATIONS	xiv
LISTE DES ANNEXES	xv
CHAPITRE 1 INTRODUCTION	1
1.1 Définitions et concepts de base	1
1.2 Éléments de la problématique	3
1.3 Objectifs de recherche	4
1.4 Plan de la thèse	5
CHAPITRE 2 REVUE DE LITTÉRATURE	6
2.1 Méthodes duales fractionnaires	6
2.1.1 Branch-and-Bound	6
2.1.2 Branch-and-Price	7
2.2 Méthodes primales	10
2.2.1 Sans décomposition	10
2.2.2 Avec décomposition	12
2.3 Apprentissage machine	16
2.3.1 Apprentissage supervisé	16
2.3.2 Apprentissage profond	17
2.3.3 Apprentissage machine et optimisation combinatoire	18
2.4 Domaines d'applications	19

2.4.1	Problème d'horaires de chauffeurs et d'autobus (VCSP)	19
2.4.2	Problème de rotations d'équipages (CPP)	19
CHAPITRE 3 ORGANISATION DU TRAVAIL		20
CHAPITRE 4 ARTICLE 1: INTEGRAL COLUMN GENERATION FOR THE SET PARTITIONING PROBLEM		22
4.1	Introduction	22
4.2	Literature review	24
4.2.1	Branch-and-price algorithms	24
4.2.2	Primal algorithms	26
4.3	The ISUD algorithm	28
4.3.1	Description of the ISUD algorithm	28
4.3.2	A small example	32
4.4	Methodology	34
4.4.1	The ICG algorithm	34
4.4.2	An acceleration strategy	39
4.5	Computational results	39
4.5.1	VCSP results	40
4.5.2	CPP results	44
4.5.3	Sensitivity analysis of the ICG parameters	47
4.6	Conclusion	48
CHAPITRE 5 ARTICLE 2: INTEGRAL COLUMN GENERATION FOR SET PAR- TITIONING PROBLEMS WITH SIDE CONSTRAINTS		50
5.1	Introduction	50
5.2	Literature review and contributions	52
5.2.1	Dual fractional algorithms	52
5.2.2	Primal algorithms	53
5.2.3	Main contributions	57
5.3	Side constraints and quasi-integrality properties	57
5.4	Handling side constraints in ISUD	60
5.4.1	Handling side constraints in the RP	60
5.4.2	Handling side constraints in the CP	61
5.4.3	Possible RP-CP combinations	65
5.5	Improved integral column generation (I ² CG)	65
5.5.1	Solving the pricing subproblem	67

5.5.2	RP-CP loop	68
5.5.3	Zooming	70
5.5.4	Control and stopping condition	71
5.6	Computational results	71
5.6.1	Performance tests	72
5.6.2	Some insights	74
5.6.3	Sensitivity of I^2CG_3 to right-hand side values	76
5.7	Conclusion	77
CHAPITRE 6 AMÉLIORATION DE LA MÉTHODE DE GÉNÉRATION DE COLONNES EN NOMBRES ENTIERS À L'AIDE DE LA PRÉDICTION DE SUCCESSIONS DE TÂCHES		79
6.1	Introduction	79
6.2	Historique de solutions	80
6.3	Modèle de prédiction	82
6.3.1	Transformation et filtrage des données	83
6.3.2	Couche cachée d'encodage	85
6.3.3	Architecture du réseau de neurones	85
6.4	Génération de colonnes en nombres entiers améliorée	86
6.5	Expérimentation	90
6.5.1	Résultats des modèles de prédiction	90
6.5.2	Résultats d' I^2CG_p	91
6.6	Conclusion	98
CHAPITRE 7 DISCUSSION GÉNÉRALE		99
CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS		101
8.1	Synthèse des travaux	101
8.2	Limitations des algorithmes proposés	102
8.3	Améliorations futures	102
RÉFÉRENCES		104
ANNEXES		108
A.1	Compatibility matrices	108
A.2	Multi-phase strategy	109
A.3	Computation of a complete dual solution	110
B.1	Detailed sensitivity results	111

LISTE DES TABLEAUX

Table 4.1	Results for the small VCSP instances	42
Table 4.2	RMH and DH results for small VCSP instances when generating many columns	43
Table 4.3	Results for the large VCSP instances	44
Table 4.4	Results of ICG with the MDCP strategy on the large VCSP instances	44
Table 4.5	Results for the CPP instances	45
Table 4.6	Sensitivity analysis results for the CPP instances	48
Table 5.1	Some feasible and infeasible solutions of model $\mathbb{P}_1$	58
Table 5.2	Results of I^2CG_2 , I^2CG_3 , DH and RMH.	73
Table 5.3	Detailed results of I^2CG_3	75
Table 5.4	Detailed results of DH and RMH.	75
Table 5.5	Sensitivity analysis average results.	77
Tableau 6.1	Les caractéristiques des vols.	82
Tableau 6.2	Résultats du filtre	84
Tableau 6.3	Les plages de valeurs des hyper-paramètres utilisés dans l'optimisation.	86
Tableau 6.4	Précision du réseau de neurones.	91
Tableau 6.5	Résultats d' I^2CG , I^2CG_p et DH pour les instances de base.	93
Tableau 6.6	Détail des résultats d' I^2CG et d' I^2CG_p pour les instances de base. . .	93
Tableau 6.7	Résultats d' I^2CG , I^2CG_p et DH pour les nouvelles instances.	95
Tableau 6.8	Détail des résultats d' I^2CG , I^2CG et DH pour les nouvelles instances.	96
Tableau B.1	Sensitivity analysis results of I^2CG_2 , I^2CG_3 , DH and RMH.	112

LISTE DES FIGURES

Figure 4.1	The three-level decomposition of the ICG algorithm	35
Figure 4.2	Current solution cost in function of the time (instance 320)	46
Figure 4.3	Number of pairings from the final solution already generated at each iteration (instance 320)	47
Figure 5.1	$\mathcal{F}_{SPPSC\mathcal{LR}}$, $conv(\mathcal{F}_{SPPSC})$ and $conv(\mathcal{F}_{SPP})$ of model $\mathbb{P}_1$	59
Figure 5.2	Cost of the current solution in function of the computational time – Instance 320.	74
Figure 5.3	Minimum reduced cost at each CG iteration for I^2CG_3 and DH – Instance 320.	76
Figure 6.1	Diagramme de flux de données.	80
Figure 6.2	Exemple d’une rotation d’un équipage.	81
Figure 6.3	Illustration d’un scénario des vols successeurs d’un vol.	81
Figure 6.4	Illustration des vols après filtrage.	84
Figure 6.5	Le coût de la solution courante en fonction du temps pour l’instance 320.	94
Figure 6.6	Le nombre de colonnes générées par itération de la solution de l’instance 320	94

LISTE DES SIGLES ET ABRÉVIATIONS

SPP	Set Partitioning Problem
SPPSC	Set Partitioning Problem with Side Constraints
CG	Column Generation
RMP	Restricted Master Problem
SP	SubProblem
IPS	Improved Primal Simplex
RP	Reduced Problem
CP	Complementary Problem
IS	Integral Simplex
ISUD	Integral Simplex Using Decomposition
I ² SUD	Improved Integral Simplex Using Decomposition
CPP	Crew Pairing Problem
VCSP	Vehicle and Crew Scheduling Problem
ICG	Integral Column Generation
I ² CG	Improved Integral Column Generation
DH	Diving Heuristic
RMH	Restricted Master Heuristic

LISTE DES ANNEXES

Annexe A		108
Annexe B		111

CHAPITRE 1 INTRODUCTION

Au fil de l'histoire, des méthodes de recherche opérationnelle ont été introduites, améliorées et utilisées pour résoudre des problèmes de la vie réelle dont la complexité dépasse la capacité des humains. Initialement, cette branche des mathématiques appliquées a été développée dans un contexte militaire afin de prendre de meilleures décisions de ravitaillement durant la guerre. Le grand succès de ces méthodes était derrière leur extension à d'autres domaines comme le commerce et l'industrie. De nos jours, les chercheurs scientifiques continuent à chercher des percées dans des disciplines (e.g., informatique) permettant de développer des méthodes de recherche opérationnelle plus efficaces. La présente thèse s'inscrit dans le cadre d'amélioration et de développement de nouvelles méthodes d'optimisation linéaires en nombres entiers pour la résolution de problèmes industriels difficiles, comme ceux de tournées de véhicules et de rotations d'équipages dans le domaine aérien (CPP : Crew Pairing Problem). Ces problèmes sont souvent formulés sous forme de problème de partitionnement d'ensemble avec des contraintes supplémentaires.

1.1 Définitions et concepts de base

Dans cette thèse, nous introduisons de nouvelles approches pour la résolution de ce type de problèmes qui consistent à couvrir un ensemble de tâches (exactement une seule fois) avec des chemins de coût minimum. Une tâche correspond à un client à visiter ou un vol d'avion à couvrir, alors qu'un chemin représente une tournée de véhicule, un horaire d'un chauffeur d'autobus ou celui d'un équipage aérien. D'autres contraintes supplémentaires, comme la disponibilité des véhicules ou celle des équipages, peuvent être ajoutées à la formulation du problème de partitionnement d'ensemble.

Nous notons par $\mathbf{T} = \{1, \dots, m\}$ l'ensemble de tâches (contraintes de partitionnement), $\mathbf{H}$ l'ensemble d'indices des contraintes supplémentaires et $\mathbf{N} = \{1, \dots, n\}$ l'ensemble des colonnes (chemins). À chaque colonne $j \in \mathbf{N}$, une variable binaire x_j est associée ; x_j prend la valeur 1 si la colonne j de coût c_j fait partie de la solution et 0 sinon. Le coefficient a_{ij} prend la valeur 1 si la colonne $j \in \mathbf{N}$ couvre la tâche $i \in \mathbf{T}$ et 0 sinon. L'ensemble des coefficients a_{ij} constitue la matrice des contraintes de partitionnement $\mathbf{A} = (A_j)_{j \in \mathbf{N}} = (a_{ij})_{i \in \mathbf{T}, j \in \mathbf{N}}$. Le coefficient $q_{hj} \in \mathbb{R}$ correspond à la contribution de la variable x_j dans la contrainte supplémentaire $h \in \mathbf{H}$, dont le membre de droite est $b_h \in \mathbb{R}$. La formulation mathématique

du problème de partitionnement d'ensemble est donnée par le modèle $\mathbb{P}$ ci-dessous :

$$\min_x \sum_{j \in \mathbf{N}} c_j x_j \quad (1.1)$$

$$(\mathbb{P}) \quad \text{sc :} \quad \sum_{j \in \mathbf{N}} a_{ij} x_j = 1, \quad \forall i \in \mathbf{T} \quad (1.2)$$

$$\sum_{j \in \mathbf{N}} q_{hj} x_j \leq b_h, \quad \forall h \in \mathbf{H} \quad (1.3)$$

$$x_j \in \{0, 1\}, \quad \forall j \in \mathbf{N}. \quad (1.4)$$

La fonction objectif (1.1) vise à minimiser le coût total de la solution. Les contraintes de partitionnement (1.2) assurent que chaque tâche soit couverte exactement une seule fois. Les contraintes supplémentaires (1.3) permettent de modéliser d'autres aspects du problème traité, par exemple, les contraintes de disponibilité d'équipages par base dans le CPP. Ces contraintes supplémentaires peuvent être des contraintes $\leq$, $=$ ou $\geq$. Enfin, des contraintes d'intégralité (1.4) sont imposées aux variables.

Dans la suite du document, nous utilisons SPP et SPPSC pour désigner respectivement le problème de partitionnement d'ensemble sans ou avec contraintes supplémentaires (1.3). Nous désignons respectivement par $\mathcal{F}_{SPP}$, $SPP^{\mathcal{LR}}$, $\mathcal{F}_{SPP\mathcal{LR}}$ et $\text{conv}(\mathcal{F}_{SPP})$ le domaine réalisable du SPP, sa relaxation continue (i.e., $x_j \geq 0, \forall j \in \mathbf{N}$), le domaine réalisable de $SPP^{\mathcal{LR}}$ et l'enveloppe convexe des points extrêmes entiers du $\mathcal{F}_{SPP}$. De la même façon, nous définissons $\mathcal{F}_{SPPSC}$, $SPPSC^{\mathcal{LR}}$, $\mathcal{F}_{SPPSC\mathcal{LR}}$ et $\text{conv}(\mathcal{F}_{SPPSC})$ pour le SPPSC.

Depuis 1969, les chercheurs ont accordé une attention particulière au SPP qui se distingue par la propriété de *quasi-intégralité*. Elle signifie que toute arête de $\text{conv}(\mathcal{F}_{SPP})$ est une arête de $\mathcal{F}_{SPP\mathcal{LR}}$. Ceci implique l'existence d'un chemin de coût décroissant (le long des arêtes du $\mathcal{F}_{SPP\mathcal{LR}}$), dit *entier*, entre chaque paire de points extrêmes entiers ; ce chemin est constitué uniquement des points extrêmes entiers. L'envie d'explorer cette propriété intéressante a donné naissance à plusieurs méthodes de résolution intéressantes, entre autres, l'algorithme du simplexe en nombres entiers (IS : Integral Simplex) [1, 2, 3], l'adaptation d'IS dans le contexte de génération de colonnes (ISCG) [4, 5] et l'algorithme du simplexe en nombres entiers avec décomposition (ISUD : Integral Simplex Using Decomposition) [6] et ses variantes (ZOOM et I²SUD) [7, 8]. Ces méthodes sont classées dans la littérature comme étant des méthodes *primales*. Elles consistent à améliorer, itérativement, une solution entière non optimale en effectuant des pivots entiers (i.e., conservent l'intégralité de la solution). Le grand avantage de ce type de méthodes est qu'une solution entière est toujours en main, ce qui permettra d'arrêter la résolution à tout moment (si les ressources temps et/ou mémoire sont épuisées).

D'autres méthodes, dites *duales*, ont été aussi développées au cours des dernières années dans

le but de résoudre le problème SPPSC (ou SPP). Elles consistent à résoudre une suite de $\text{SPPSC}^{\mathcal{LR}}$ (ou $\text{SPP}^{\mathcal{LR}}$) en maintenant l’optimalité de la fonction objectif (1.1) et la faisabilité des contraintes linéaires (1.2)-(1.3), jusqu’à atteindre l’intégralité. Parmi ces méthodes, on trouve la méthode d’énumération implicite *Branch-and-Bound* [9] qui cherche des solutions entières dans un arbre de branchement lorsque la solution du $\text{SPPSC}^{\mathcal{LR}}$ est fractionnaire. Dans chacun de ses nœuds de branchement, on résout un $\text{SPPSC}^{\mathcal{LR}}$ incluant les contraintes associées aux décisions de branchement qui ont mené à ce nœud. Lorsque les colonnes de la matrice des contraintes ne sont pas connues a priori, on parle dans ce cas de *Branch-and-Price* [10]. C’est une méthode de *Branch-and-Bound*, où la méthode de génération de colonnes (CG : Column Generation) est utilisée pour résoudre le $\text{SPPSC}^{\mathcal{LR}}$ à chaque nœud de branchement. En effet, les colonnes qui ne sont pas générées dans le nœud racine de l’arbre de *Branch-and-Bound* peuvent être nécessaires pour obtenir une solution entière optimale. La CG est une méthode populaire permettant de résoudre diverses classes de problèmes de décision d’intérêt pratique, entre autres, les problèmes de tournées de véhicules et de rotations d’équipages de grande taille. La CG est jumelée à la décomposition de Danzinger-Wolfe [11] ; qui consiste à décomposer le problème traité en un problème maître restreint (RMP : Restricted Master Problem), souvent résolu avec la méthode du simplexe et un sous-problème (SP : SubProblem), souvent formulé comme un problème de plus court chemin avec contraintes de ressources qui est résolu par le biais de la programmation dynamique. Notons qu’il est possible d’avoir plusieurs SPs. À chaque itération de CG, des colonnes prometteuses (i.e., de coût réduit négatif pour un problème de minimisation) sont ajoutées au RMP. Ces colonnes sont obtenues en résolvant un ou plusieurs SPs. L’approche est basée sur l’observation suivante : pour les grands problèmes, la majorité des colonnes ne sont pas nécessaires pour obtenir une solution optimale, c’est-à-dire que leurs variables correspondantes y seront égales à zéro (dans cette solution optimale), c’est pour cette raison qu’on parle de RMP. La nécessité de résoudre efficacement les grands problèmes de tournées de véhicules et de rotations d’équipages a conduit naturellement au développement de nombreuses méthodes d’optimisation basées sur la CG.

1.2 Éléments de la problématique

Outre leurs avantages, les méthodes primales et duales introduites précédemment présentent certains inconvénients. D’une part, les algorithmes IS et ISCG, hautement combinatoires, ne sont pas efficaces pour les grands problèmes, principalement à cause de la dégénérescence sévère, qui signifie que le nombre des bases dégénérées adjacentes associées à un même point extrême pourrait être énorme. Ceci implique une difficulté à trouver des pivots entiers pour

améliorer la solution entière courante. Ce problème a été surmonté suite à l'introduction du nouvel algorithme ISUD et ses variantes ZOOM et I²SUD. Ces derniers trouvent une suite de solutions entières menant à une solution optimale ou quasi-optimale. Ceci en décomposant le SPP en deux sous-problèmes permettant de trouver un sous-ensemble de variables à pivoter dans la base afin d'améliorer la solution entière courante. Malgré leur efficacité, ces trois algorithmes ont deux limitations : i) ils supposent que les colonnes de la matrice des contraintes sont générées a priori. Ceci n'est généralement pas possible pour les grands problèmes. ii) Aucun de ces algorithmes ne gère les contraintes supplémentaires (1.3). Ces dernières sont difficiles à gérer car elles brisent la propriété de quasi-intégralité du SPP. Donc, les contraintes supplémentaires accentuent la difficulté de résolution du SPP, en particulier pour les méthodes primales car elles peuvent être coincées dans un minimum local lorsque les contraintes supplémentaires coupent les chemins entiers qui mènent de la solution courante à une solution optimale.

D'autre part, pour certains types de problèmes, les méthodes duales (e.g., *Branch-and-Price*) trouveraient facilement une solution entière de bonne qualité en utilisant la stratégie de branchement en profondeur d'abord ; dans la pratique, ce n'est souvent pas le cas. En effet, à chaque itération de CG, plusieurs colonnes de coût réduit négatif sont générées. Les colonnes qui sont bonnes pour la relaxation continue sont gardées, et d'autres qui favorisent l'intégralité de la solution peuvent être ignorées. Ces dernières vont être probablement générées à nouveau dans un autre nœud de branchement, ce qui implique un grand nombre d'itérations de CG et l'augmentation du temps de résolution du SP. Ce dernier est souvent difficile à résoudre à cause du nombre de ressources, généralement élevé (≥ 4), et la cyclicité du réseau. De plus, la CG souffre de la dégénérescence inhérente au SPP, qui se manifeste quand le RMP est résolu avec la méthode de simplexe. Théoriquement, ce phénomène se produit si au moins une variable de base est nulle ; mais dans la pratique, la dégénérescence est généralement sévère, c'est-à-dire que plusieurs variables de base sont nulles. Pour contourner la dégénérescence, la méthode de points intérieurs (Cplex Barrier) est souvent employée. Les solutions trouvées par cette méthode sont habituellement très fractionnaires, ce qui signifie que la taille de l'arbre de branchement peut augmenter drastiquement, et qu'une bonne solution entière ne peut être trouvée qu'à la fin de la résolution.

1.3 Objectifs de recherche

L'objectif général de cette thèse est de proposer de nouvelles méthodes primales de génération de colonnes en nombres entiers capables de résoudre efficacement les SPPs et SPPSCs de grande taille tout en surmontant certaines limitations des autres méthodes primales et duales.

Dans un premier temps, nous exploitons les résultats théoriques précédemment développés dans l'article [6] pour proposer une nouvelle méthode de génération de colonnes en nombres entiers, nommée ICG. Dans cette méthode, nous adaptons l'algorithme ISUD au contexte de génération de colonnes. Le but est de générer une suite de solutions entières, de coût décroissant, menant à une solution optimale ou quasi-optimale. Ensuite, nous proposons une version améliorée d'ICG, baptisée I²CG, permettant de résoudre des SPPSCs. L'amélioration concerne principalement la gestion des contraintes supplémentaires au niveau du RMP. Pour cela, nous montrons théoriquement que les contraintes supplémentaires brisent la propriété de quasi-intégralité du SPP. Afin de restaurer les conditions favorables aux méthodes primales, nous proposons d'ajouter des arêtes à $\mathcal{F}_{SPPSC\mathcal{L}\mathcal{R}}$ en augmentant sa dimension pour créer de nouveaux chemins entiers qui remplacent ceux coupés à cause des contraintes supplémentaires. Finalement, nous proposons une version améliorée de la méthode I²CG, basée sur les résultats d'un modèle de prédiction d'apprentissage profond, pour résoudre efficacement le CPP. Ce modèle de prédiction est entraîné sur un jeu de données pour prédire avec une certaine probabilité les connexions (successions) entre les vols. Les probabilités résultantes sont utilisées pour accélérer la génération des rotations ayant une grande probabilité d'appartenir à une solution entière optimale ou quasi-optimale.

1.4 Plan de la thèse

Le présent document est organisé comme suit : Le chapitre 2 est dédié à la revue de littérature sur les méthodes duales, les méthodes primales, l'apprentissage machine et les domaines d'application de nos méthodes. Le chapitre 3 discute brièvement l'organisation des trois chapitres principaux de la thèse. Le chapitre 4 décrit la méthode de génération de colonnes en nombres entiers (ICG) développée dans le cadre du premier sujet de la thèse. Il s'agit d'un article publié dans la revue EURO Journal on Transportation and Logistics. Le chapitre 5 comporte le texte du deuxième article, soumis à la revue INFORMS Journal on Computing, dans lequel nous présentons la méthode I²CG. Le chapitre 6 présente l'amélioration de la méthode I²CG. Une discussion générale et une conclusion sont présentées respectivement dans les chapitres 7 et 8.

CHAPITRE 2 REVUE DE LITTÉRATURE

Les problèmes de types partitionnement d'ensemble sont généralement résolus à l'aide de trois classes de méthodes (Letchford et Lodi [12]) : duales entières, duales fractionnaires et primales. La première classe de méthodes consiste à maintenir l'intégralité et l'optimalité tant que les contraintes linéaires primales ne sont pas satisfaites. Étant donné que ce type de méthodes est très peu utilisé, on se contente alors de présenter une revue de littérature sur les deux autres classes de méthodes ainsi que les algorithmes d'apprentissage machine et les domaines dans lesquelles nous avons appliqué nos méthodes.

2.1 Méthodes duales fractionnaires

Ces méthodes consistent à relaxer les contraintes d'intégralité des variables (1.4) tout en maintenant à chaque itération, l'optimalité de la fonction objectif (1.1) et la faisabilité des contraintes linéaires (1.2)-(1.3). Parmi ces méthodes, on trouve *Branch-and-Bound* (Morrison et al. [9]) et *Branch-and-Price* (Barnhart et al. [10]).

2.1.1 Branch-and-Bound

Branch-and-Bound est un algorithme exact permettant de résoudre les programmes linéaires en nombres entiers dont les colonnes de la matrice des contraintes sont connues à l'avance. Il explore un arbre de branchement pour énumérer implicitement les solutions possibles du problème traité. Le processus de branchement est déclenché lorsque la solution optimale du $\mathcal{F}_{SPPSC\mathcal{LR}}$, au nœud racine, est fractionnaire. Dans ce cas, une stratégie d'exploration (e.g., profondeur d'abord) est appliquée pour choisir un nœud k de branchement dont la solution de la relaxation linéaire associée est fractionnaire. Ensuite, une règle de branchement, dite aussi règle de séparation est utilisée pour créer les nœuds fils du nœud k . Dans chacun des nœuds ainsi créés, une coupe résultante de la règle de séparation (e.g., fixer une variable à 0 ou à 1) est ajoutée à la relaxation linéaire correspondante pour couper la solution fractionnaire trouvée dans le nœud k . Les solutions des relaxations linéaires (donnant des bornes inférieures) ainsi que les solutions entières (donnant des bornes supérieures) rencontrées au cours de l'exploration de l'arbre sont utilisées pour élaguer les nœuds. Ce processus de branchement est répété tant qu'il reste des branches non exploitées.

L'article [9] présente une revue de littérature détaillée sur les différentes stratégies d'exploration, règles de branchement et règles d'élagage des nœuds utilisées dans l'algorithme

Branch-and-Bound.

2.1.2 Branch-and-Price

Branch-and-Price est une méthode populaire et efficace permettant de résoudre des problèmes industriels de grande taille. Elle consiste à intégrer la génération de colonnes dans un processus de *Branch-and-Bound*, où chaque nœud k de l'arbre de branchement est résolu à l'optimalité avec CG. Cette dernière est une méthode exacte qui résout efficacement la relaxation continue des programmes linéaires de grande taille, souvent décomposés en RMP et SP en utilisant la décomposition de Dantzig-Wolfe [11]. Une description des algorithmes génériques de *Branch-and-Price* est présentée par Barnhart et *al.* [10].

Décomposition de Dantzig-Wolfe.

La décomposition de Dantzig-Wolfe [11] permet de résoudre efficacement des programmes linéaires, de très grande taille, ayant une structure particulière (une partie de la matrice des contraintes est diagonale par blocs). Cette méthode consiste à décomposer le problème original en un problème maître (MP : Master Problem) et un ou plusieurs SPs. Le MP est une reformulation du problème original en fonction des points extrêmes des SPs. Ces points extrêmes sont liés par un sous-ensemble de contraintes du problème original, dites contraintes liantes. Le SP comporte les blocs de contraintes qui ne sont pas liantes (i.e., qui concernent des groupes de variables distincts). Plusieurs SPs peuvent être créés si la matrice de contraintes du problème original contient des blocs indépendants. Le programme ainsi reformulé présente moins de contraintes, mais beaucoup plus de variables. Cette difficulté peut être contournée en faisant appel à des techniques de génération de colonnes pour générer des points extrêmes et des rayons extrêmes du domaine réalisable des SPs au fur et à mesure de la résolution.

L'efficacité de la décomposition de Dantzig-Wolfe a été illustrée par Gilmore et Gomory [13] en l'appliquant au problème de découpe. Cette décomposition permet d'avoir une relaxation serrée des programmes linéaires en nombres entiers si une partie de l'intégralité de la solution est gérée au niveau du SP. Cette propriété a été exploitée par Desrosiers et *al.* [14] et d'autres (voir [15]) pour résoudre, à l'optimalité, des problèmes en nombres entiers par CG.

Génération de colonnes (CG).

La CG se base sur la résolution itérative du RMP et du SP. Ce dernier est défini avec les variables duales des contraintes du RMP. À chaque itération, de nouveaux points extrêmes du SP sont générés et ajoutés au RMP qui est ré-optimisé par la suite. Si aucune colonne

n'est générée, alors on peut dire que la solution courante du RMP est optimale et qu'une solution optimale est trouvée pour le problème original.

Formellement, soit le MP formulé à l'aide de la relaxation linéaire du modèle $\mathbb{P}$. Pour résoudre ce MP, nous cherchons une combinaison linéaire, de points extrêmes du domaine réalisable du SP, qui minimise la fonction objectif (1.1) et satisfait les contraintes (1.2)-(1.3). Puisque le nombre des points extrêmes du SP ($|\mathbf{N}|$) est généralement exponentiel en fonction du nombre de tâches, on se limite alors à un sous-ensemble de points extrêmes du SP $\tilde{\mathbf{N}} \subset \mathbf{N}$. Soit $\pi \in \mathbb{R}^{|\mathbf{T}|+|\mathbf{H}|}$ le vecteur des variables duales associées à (1.2) et (1.3). L'objectif du SP est défini par :

$$\bar{c}^* = \min_{j \in \mathbf{N}} (\bar{c}_j = c_j - \sum_{i \in \mathbf{T}} \pi_i a_{ij} - \sum_{h \in \mathbf{H}} \pi_h q_{hj}) \quad (2.1)$$

À chaque itération de CG, un ou plusieurs SPs de fonction objectif (2.1) sont résolus pour générer des colonnes de coût réduit $\bar{c}_j$ négatif, qui peuvent entrer dans la base et réduire la valeur de l'objectif du RMP. La résolution du RMP à l'aide de CG nous donne une solution optimale x^* et duale π^* telles que :

$$\bar{c}_j \geq 0, \quad \forall j \in \mathbf{N} \quad (2.2)$$

$$x_j^* = 0, \quad \forall j \in \mathbf{N} \setminus \tilde{\mathbf{N}}. \quad (2.3)$$

Les inéquations (2.2) sont les conditions d'arrêt de l'algorithme de CG. Les égalités (2.3) indiquent que toutes les variables non considérées dans le RMP sont égales à zéro. À partir de ces conditions, on peut dire que la solution primale duale est optimale pour le MP.

Malgré son efficacité, la méthode de CG est influencée négativement par plusieurs facteurs, liés principalement à la dégénérescence, qui ralentissent sa convergence (voir [16]). Ces problèmes de convergence de CG impliquent un problème de convergence de *Branch-and-Price*. Dans la littérature, plusieurs techniques de stabilisation de CG ont été introduites (voir [15, 16]). Entre autres, l'utilisation de la méthode de points intérieurs, à la place de la méthode du simplexe, permet d'avoir une solution duale "centrale" au lieu d'une solution duale "extrême" correspondante à une solution de base dégénérée. Cependant, la méthode de points intérieurs favorise les solutions primales fractionnaires, ce qui peut faire exploser l'arbre de branchement et ralentit la convergence de *Branch-and-Price*. D'autres travaux de recherche [17, 18, 19, 20] ont aussi été menés pour remédier aux problèmes liés à la convergence.

En 2005, Elhallaoui et *al.* [17] ont introduit la méthode DCA. Elle consiste à réduire le nombre de contraintes de partitionnement du RMP en regroupant certaines d'entre elles. Les

résultats expérimentaux montrent que l'approche DCA surpasse clairement la méthode de génération de colonnes standard pour la résolution du SPP relaxé. Dans leurs expériences sur de grandes instances du problème d'horaires de chauffeurs et d'autobus (VCSP : vehicle and crew scheduling problem), l'algorithme DCA a réduit le temps de résolution par 80%. Plus tard, Elhallaoui et *al.* [18] ont présenté une version améliorée de DCA, dite méthode d'agrégation dynamique des contraintes avec la stratégie multi-phase (MPDCA). Dans cette méthode, la stratégie *partial pricing* est utilisée pour favoriser les colonnes qui sont *compatibles* ou peu *incompatibles* avec l'agrégation de contraintes courante.

Définition 1. *Un sous-ensemble $\mathcal{U}$ de $\mathbf{N}$ est dit compatible avec un ensemble de colonnes indexées dans $\mathcal{S}$, ou simplement compatible, s'il existe deux vecteurs $v \in \mathbb{R}_+^{|\mathcal{U}|}$ et $\lambda \in \mathbb{R}^{|\mathcal{S}|}$ tels que $\sum_{j \in \mathcal{U}} v_j A_j = \sum_{l \in \mathcal{S}} \lambda_l A_l$. La combinaison de colonnes, éventuellement un singleton, $\sum_{j \in \mathcal{U}} v_j A_j$ est dite compatible.*

La stratégie *partial pricing* permet de i) maintenir un niveau supérieur d'agrégation ; ii) réduire encore la dégénérescence ; iii) minimiser le nombre de variables fractionnaires dans les solutions obtenues. Par conséquent, MPDCA trouve fréquemment des solutions entières. DCA réduit aussi le temps de résolution de la relaxation linéaire, mais elle n'élabore pas les directions de descente. Autrement dit, la méthode DCA se comporte comme CG standard en pivotant sur une seule variable à la fois. Ceci est considéré comme une limite de la méthode.

En 2011, Elhallaoui et *al.* [19] ont introduit l'algorithme du simplexe primal amélioré (IPS : Improved Primal Simplex). C'est un algorithme de résolution d'un programme linéaire dégénéré quelconque. IPS consiste à décomposer le problème en deux sous-problèmes : un problème réduit (RP : Reduced Problem) contenant uniquement les colonnes compatibles avec la solution courante $\mathcal{S}$ et un problème complémentaire (CP : Complementary Problem) contenant les colonnes incompatibles avec la solution $\mathcal{S}$. Formellement, une colonne est dite compatible si elle peut s'écrire comme combinaison linéaire des colonnes de la solution, sinon elle est dite incompatible. Le RP améliore la solution $\mathcal{S}$ en pivotant sur une variable de coût réduit négatif correspondant à une colonne compatible, tandis que le CP cherche une combinaison compatible (avec $\mathcal{S}$) de colonnes incompatibles. Cette combinaison devrait améliorer, après échange, le coût de la solution courante. Le processus s'arrête lorsque ni le RP ni le CP ne peuvent améliorer la solution courante. Les résultats numériques montrent que IPS permet d'augmenter l'efficacité de la méthode du simplexe face à la dégénérescence. En combinant IPS [19] et la CG, Bouarab et *al.* [20] montrent que IPS permet de résoudre efficacement le RMP, où un changement de variables a permis de réduire le nombre d'éléments non nuls dans la matrice de contraintes du CP. Par conséquent, plus de variables ont été considérées, ce qui a stabilisé les variables duales et réduit le nombre d'itérations de CG. Comme DCA,

IPS réduit considérablement la dégénérescence et le temps de résolution. Plus encore, IPS considère un grand nombre de colonnes par rapport à CG standard, et trouve les directions de descente pour améliorer le RMP. Toutefois, IPS se limite seulement à la résolution de la relaxation continue et ne traite pas les programmes linéaires en nombres entiers.

2.2 Méthodes primales

À l'encontre des méthodes duales, les méthodes primales maintiennent la faisabilité des contraintes linéaires (1.2)-(1.3) et d'intégralité des variables (1.4). Elles s'arrêtent lorsque l'optimalité est atteinte. Autrement dit, ces méthodes cherchent une suite non croissante de solutions entières menant à la solution optimale. Dans cette section, nous présentons une revue de littérature sur deux types de méthodes primales, à savoir les méthodes primales sans décomposition du SPP et les méthodes primales avec décomposition.

2.2.1 Sans décomposition

En 1969, Trubin [21] a montré que les arêtes de $\text{conv}(\mathcal{F}_{SPP})$ sont aussi des arêtes de $\mathcal{F}_{SPP\mathbb{Z}}$, cette propriété a été nommée *quasi-intégralité*. Elle implique l'existence d'un chemin reliant toute paire de points extrêmes entiers (voir Balas et Padberg [22]). Ce chemin passe seulement par des solutions entières de coût décroissant. Balas et Padberg [22] ont aussi montré que la solution optimale peut être atteinte en au plus m pivots. Cependant ce résultat n'est pas utile dans la pratique car il nécessite la connaissance d'une solution optimale.

En 2002, Thompson [2] a introduit la méthode du simplexe en nombres entiers en se basant sur les résultats de Balas et Padberg [22]. IS est composé de deux méthodes : méthode locale et méthode globale. La méthode locale consiste à chercher un optimum local en effectuant, tant que c'est possible, des pivots sur 1 (voir définition 2 dans l'article [4]). La méthode globale consiste à construire un arbre de branchement dont chaque nœud correspond à un sous-problème qu'on résout avec la méthode locale. La solution optimale du problème original est la meilleure de toutes les solutions obtenues durant la résolution des nœuds de branchement. Une amélioration de la méthode de Thompson a été apportée par Saxena [3], en introduisant des pivots dégénérés sur -1 et en adoptant des règles d'anti-cyclage. Saxena a démontré qu'une solution optimale peut être atteinte en n'utilisant que des pivots sur 1 dans la méthode de Thompson. La méthode IS assure qu'une solution entière est toujours en main, ce qui permettra d'arrêter la résolution à tout moment. Cependant, cette méthode souffre aussi de la dégénérescence et la difficulté de trouver un pivot non dégénéré pour améliorer la solution entière courante.

En 2009, Rönnberg et Larsson [4] ont proposé une adaptation de la méthode IS à la génération de colonnes pour résoudre le SPP. Dans un premier temps, Rönnberg et Larsson ont proposé une méthode, nommée ISCG, qui permet de générer et identifier des variables permettant des pivots non-dégénérés et dégénérés. Pour ce faire, les auteurs ont adapté la condition nécessaire (de Balas et Padberg [22]) pour qu'un pivot soit non-dégénéré. Cette condition a été ajoutée par la suite au SP (programme linéaire) pour générer des variables permettant des pivots non-dégénérés. De la même façon, la condition sur les pivots dégénérés a été ajoutée au SP pour générer d'autres variables permettant des pivots dégénérés. La suite des pivots non-dégénérés et dégénérés trouvés à l'aide de la méthode ISCG n'assure qu'un optimum local. Donc, pour avoir un optimum global, les auteurs proposent d'explorer un arbre de branchement où chaque nœud de branchement correspond à un SPP (avec des variables fixées à 1 ou 0) qui est résolu à l'aide d'ISCG. Cependant, cette méthode n'est pas utile dans la pratique. D'un côté, la CG non-dégénérée ne génère que les colonnes compatibles avec la solution entière courante. Ces colonnes sont rares dans la pratique. D'un autre côté, la CG dégénérée peut impliquer une perte de temps de résolution sans améliorer la solution de base. Des exemples académiques ont été utilisés pour illustrer quelques propos, mais aucune expérimentation n'a été effectuée sur de vraies instances.

En 2014, Rönnberg et Larsson [5] ont proposé une nouvelle version d'ISCG en ajoutant une variable correspondante à une direction de descente dans le tableau du simplexe. Le but est de construire une variable qui permet un pivot entier non-dégénéré. Ensuite, ils ont introduit une condition d'optimalité pour vérifier si la solution optimale du SPP original est contenue dans le RMP. Cette solution optimale peut être trouvée en utilisant une stratégie de branchement adaptée à l'IS (voir Thompson [2]). Malgré les améliorations apportées à leur méthode, elle reste loin d'être performante car la CG non-dégénérée et la CG dégénérée restent des éléments clés dans l'approche malgré leur inefficacité. De plus, aucune méthode n'a été proposée pour trouver les directions de descente. Des tests ont été effectués sur cinq petites instances (10 agents et 30 tâches) du problème d'affectation car la résolution de grandes instances a été impossible à cause de l'inefficacité de la méthode proposée.

En résumé, on peut dire que le travail de Rönnberg et Larsson [4, 5] a été une tentative pour profiter de la structure particulière du SPP afin de proposer une nouvelle méthode de résolution basée sur IS et CG. Ce travail a donné quelques résultats théoriques, mais aucun résultat pratique.

2.2.2 Avec décomposition

En 2014, Zaghrouti et *al.* [6] ont proposé un nouvel algorithme primal, nommé ISUD. Ce dernier est une adaptation de l'algorithme IPS pour résoudre le SPP en le décomposant en un RP et CP. Le RP se formule comme un SPP restreint, défini à partir des variables compatibles et un sous-ensemble de contraintes du problème original. Il cherche une solution améliorée dans le sous-espace vectoriel des colonnes correspondant aux variables de base non-dégénérées. Le RP améliore la solution si au moins une de ses variables a un coût réduit négatif. Le CP est un programme linéaire construit à l'aide des colonnes incompatibles de la matrice des contraintes. Il cherche des directions de descente entières (menant à des solutions entières) dans le sous-espace vectoriel complémentaire.

Comme les autres méthodes primales, ISUD commence avec une solution entière x^0 . On note par $\mathcal{S}$, l'ensemble des indices des colonnes du support de la solution x^0 (i.e., $\mathcal{S} = \text{supp}(x^0) = \{j \in \mathbf{N} | x_j^0 \neq 0\}$). Dans la suite du document, nous disons, par abus de langage, que $\mathcal{S}$ est la solution x^0 et que les colonnes A_j , $j \in \mathcal{S}$ sont dans la solution x^0 ou dans $\mathcal{S}$. Les autres colonnes A_j , $j \in \mathbf{N} \setminus \mathcal{S}$ sont soit compatibles ou incompatibles. Soit $C_{\mathcal{S}}$ l'ensemble d'indices des colonnes compatibles (incluant celles de $\mathcal{S}$) et $I_{\mathcal{S}}$ l'ensemble d'indices des colonnes incompatibles. Notons que $I_{\mathcal{S}} \cap C_{\mathcal{S}} = \emptyset$. Les colonnes compatibles sont considérées dans le RP, dont la formulation mathématique est la suivante :

$$\min_x \quad \sum_{j \in C_{\mathcal{S}}} c_j x_j \quad (2.4)$$

$$(RP1) \quad \text{sc} : \quad \sum_{j \in C_{\mathcal{S}}} a_{ij} x_j = 1, \quad \forall i \in \tilde{\mathbf{T}} \quad (2.5)$$

$$x_j \in \{0, 1\}, \quad \forall j \in C_{\mathcal{S}} \quad (2.6)$$

où $\tilde{\mathbf{T}} \subseteq \mathbf{T}$ est l'ensemble des indices des lignes de la matrice de contraintes de RP1 linéairement indépendantes (e.g., indice de la première ligne couverte par chaque colonne A_j , $j \in \mathcal{S}$). Le RP1 peut être résolu facilement en utilisant les techniques de programmation en nombres entiers offertes par les solveurs commerciaux (e.g., Cplex). Il est à noter que RP1 améliore la solution x^0 si une variable x_j , $j \in C_{\mathcal{S}}$ a un coût réduit négatif. Dans ce cas, les ensembles $\mathcal{S}$ et $C_{\mathcal{S}}$ sont mis à jour. Ce processus peut être répété tant que le RP1 trouve une solution améliorée. Dans le cas échéant, on résout le CP pour trouver une direction de descente minimale (non-décomposable) d selon la définition suivante.

Définition 2. Soit $\mathcal{U} = \{j \in \mathbf{N} | d_j > 0\}$ l'ensemble d'indices des variables entrantes dans la direction d (vérifiant $Ad = 0$). Cette direction est dite minimale par rapport à $\mathcal{S}$ ou tout simplement minimale si et seulement si tout sous-ensemble strict de $\mathcal{U}$ est incompatible avec

$\mathcal{S}$.

Une direction de descente d est dite entière si et seulement si $x^0 + d$ est une solution entière ; sinon elle est dite fractionnaire. Algébriquement, d est entière si la combinaison des colonnes incompatibles A_j , $j \in \{j \in \mathbf{N} | d_j > 0\}$ est compatible avec $\mathcal{S}$ et composée de colonnes disjointes selon la définition suivante :

Définition 3. Soit $\mathcal{U} \subset \mathbf{N}$ un sous-ensemble d'indices de colonnes. Les colonnes A_j , $j \in \mathcal{U}$, sont dites disjointes si et seulement si $A_{j_1}^T A_{j_2} = 0 \ \forall j_1, j_2 \in \mathcal{U}, j_1 \neq j_2$.

Soient v_j , $j \in I_{\mathcal{S}}$, les variables définissant les poids des colonnes incompatibles dans la combinaison linéaire $\sum_{j \in I_{\mathcal{S}}} v_j A_j$. Soient λ_l , $l \in \mathcal{S}$, les variables définissant les poids des colonnes de $\mathcal{S}$ dans la combinaison linéaire $\sum_{l \in \mathcal{S}} \lambda_l A_l$. Soit $F_{\mathcal{S}} = \{(j_1, j_2) \in I_{\mathcal{S}} \times I_{\mathcal{S}} | A_{j_1}^T A_{j_2} \neq 0\}$ l'ensemble des paires d'indices de colonnes incompatibles non disjointes. Le problème complémentaire, noté CP1 (le nombre 1 pour signifier la première version du CP), est formulé comme suit :

$$z^{CP1} = \min_{v, \lambda} \quad \sum_{j \in I_{\mathcal{S}}} c_j v_j - \sum_{l \in \mathcal{S}} c_l \lambda_l \quad (2.7)$$

$$(CP1) \quad \text{sc :} \quad \sum_{j \in I_{\mathcal{S}}} a_{ij} v_j - \sum_{l \in \mathcal{S}} a_{il} \lambda_l = 0, \ \forall i \in T \quad (2.8)$$

$$\sum_{j \in I_{\mathcal{S}}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l \lambda_l = 1 \quad (2.9)$$

$$v_j \geq 0, \ \forall j \in I_{\mathcal{S}} \quad (2.10)$$

$$v_{j_1} v_{j_2} = 0, \ \forall (j_1, j_2) \in F_{\mathcal{S}}. \quad (2.11)$$

La fonction objectif (2.7) vise à minimiser le coût réduit. Les contraintes (2.8) assurent que la combinaison des colonnes incompatibles A_j , $j \in \{j | v_j > 0\}$ soit compatible. Les scalaires α_j , $j \in I_{\mathcal{S}}$ et β_l , $l \in \mathcal{S}$ sont les poids des variables dans la contrainte de normalisation (2.9) qui borne le problème (sans quoi, le CP1 pourrait être non borné). Les contraintes (2.11) assurent que les colonnes qui composent la direction de descente soient disjointes (deux à deux). À l'optimalité, les variables $v_j \neq 0$ désignent les variables entrantes dans la base correspondante à la solution courante, tandis que les variables $\lambda_l \neq 0$ désignent les variables sortantes de cette base. Si $z^{CP1} < 0$, alors une direction de descente entière est trouvée et la solution $\mathcal{S}$ est améliorée. Il est à noter qu'en pratique, on relâche les contraintes (2.11). Comme le montre Zaghrouti et al. [6], le CP1 relaxé trouve souvent des directions de descente entières. Lorsque ce n'est pas le cas, un branchement peut être effectué pour couper la direction fractionnaire. Par exemple, on impose $v_j = 0$ tel que $v_j > 0$ dans la direction fractionnaire.

La résolution itérative de RP1 et CP1 permet à ISUD de trouver une suite de solutions entières, de coûts non croissants, menant à une solution optimale ou quasi-optimale. Les expérimentations effectuées par Zaghroui et *al.* [6] sur 90 instances du VCSP et du CPP impliquant jusqu'à 1600 contraintes et 500000 variables (générées a priori), ont montré l'efficacité de l'algorithme ISUD qui favorise intrinsèquement l'intégralité des solutions trouvées. ISUD trouve facilement les combinaisons de colonnes recherchées par Balas et les autres (voir [1, 2, 4, 5]).

Malgré ces avantages, ISUD souffre de deux limitations majeures : i) il trouve non seulement des directions entières, mais aussi des fractionnaires (20% à 50% de directions sont fractionnaires). ii) L'amélioration de la solution par itération est plutôt petite car ISUD n'atteint que les points extrêmes adjacents au point extrême entier courant. Pour remédier à ces difficultés, des recherches ont permis d'améliorer la performance de cette méthode prometteuse. Entre autres, on trouve les travaux de Rosat et *al.* [23, 24] et Zaghroui et *al.* [7, 8].

D'une part, Rosat et *al.* [23] ont utilisé des plans coupants dans le problème complémentaire pour éliminer non seulement le point extrême fractionnaire trouvé suite à la résolution du problème CP, mais aussi la direction qui mène à ce point. Ils ont montré qu'il existe toujours un hyperplan séparant une direction fractionnaire d^f des directions qui mènent à des solutions entières. Des méthodes de construction de plans coupants ont été proposées. Cependant, ces plans coupants peuvent être difficiles à trouver et doivent être appliqués plusieurs fois pour avoir une direction de descente entière. Pour pallier ces difficultés, Rosat et *al.* [24] ont montré l'existence d'un vecteur de poids $\begin{bmatrix} \alpha & \beta \end{bmatrix}$ associé à la contrainte de normalisation, qui permet de ne trouver que des directions entières, jusqu'à atteindre l'optimalité du SPP. Malheureusement, aucune méthode de construction de tel vecteur de poids n'a été proposée. Des tests ont été effectués sur des instances de VCSP (allant jusqu'à 1600 contraintes et 570000 variables) en utilisant différentes versions de la contrainte de normalisation. Ces tests ont montré que la contrainte de normalisation ($\sum_{j \in I_S} v_j + \sum_{l \in S} \lambda_l = 1$) utilisée dans la version de base d'ISUD n'est pas la plus efficace, et que le comportement de CP est influencé par le choix de la contrainte de normalisation.

D'autre part, Zaghroui et *al.* [7] ont proposé une nouvelle approche basée sur ISUD appelée *Zooming approach* ou ZOOM pour surmonter les difficultés de la version de base d'ISUD. L'idée est d'explorer le voisinage d'une direction de descente fractionnaire d^f pour trouver une direction de descente entière. En fait, au lieu de couper ou de brancher dans le CP pour éliminer la direction d^f , cette dernière est utilisée pour guider la recherche dans une zone d'amélioration potentielle. Concrètement, un très petit SPP est défini par le voisinage de la direction d^f pour trouver une solution entière améliorée. Ce SPP réduit est résolu efficacement

par un solveur commercial (e.g., Cplex). D'après les résultats présentés, nous remarquons que ZOOM surpasse ISUD. En effet, il trouve de meilleures solutions et il est légèrement plus rapide dans la plupart des cas.

Dans la même perspective d'amélioration de l'algorithme ISUD, Zaghrouti et *al.* [8] ont proposé aussi une deuxième variante d'ISUD, nommée I²SUD, où ils ont ajouté une variable artificielle au SPP. Soit $\mathbb{K}$ un SPP et $\mathbb{K}'$ le nouveau SPP créé à partir de $\mathbb{K}$ en lui ajoutant la variable artificielle x_{n+1} , tel que $x' = \begin{bmatrix} x & x_{n+1} \end{bmatrix}$, $c' = \begin{bmatrix} c & c_L \end{bmatrix}$, $\mathbf{A}' = \begin{bmatrix} \mathbf{A} & e \end{bmatrix}$ et $\mathbf{N}' = \mathbf{N} \cup \{n+1\}$. Le coût c_L de la variable ajoutée x_{n+1} correspond au coût de la solution courante ($c_L = \sum_{l \in \mathcal{S}} c_l$). La colonne e , dont les composantes sont égales à 1, est le résultat de l'agrégation des colonnes de $\mathcal{S}$. Par construction, $\mathcal{S}' = \{n+1\}$ est une solution artificielle de $\mathbb{K}'$. Le CP du modèle $\mathbb{K}'$ est le suivant :

$$\min_{v, \lambda} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} c_j v_j + \sum_{l \in \mathcal{S}} c_l v_l - c_L \lambda \quad (2.12)$$

$$(CP2) \quad \text{sc :} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} a_{ij} v_j + \sum_{l \in \mathcal{S}} a_{il} v_l - \lambda = 0, \quad \forall i \in \mathbf{T} \quad (2.13)$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l v_l = 1 \quad (2.14)$$

$$v \geq 0, \lambda \geq 0. \quad (2.15)$$

Les variables $v_j > 0$, $j \in \mathbf{N} \setminus \mathcal{S}$ sont les variables entrantes dans la base. Les variables $v_l = 0$, $l \in \mathcal{S}$ sont les variables sortantes de la base. Les scalaires α_j et β_l représentent les poids des variables dans la contrainte de normalisation. Les directions trouvées par le CP2 sont minimales par rapport à la solution artificielle. Ceci veut dire que dans $\mathbb{K}'$, tous les points extrêmes sont adjacents à cette solution (voir la proposition 3.2 dans [8]). Donc, CP2 peut trouver une direction de descente entière qui mène directement à la solution optimale du $\mathbb{K}$. À l'encontre du CP1, les directions de descente trouvées par CP2 sont généralement non minimales par rapport à la solution courante. Donc, les directions fractionnaires peuvent contenir une combinaison de directions de descente minimales entières. Les résultats numériques ont montré que I²SUD surpasse ISUD et parvient souvent à trouver une solution optimale en résolvant un seul CP2. Les trois algorithmes ISUD, ZOOM et I²SUD ont été testés sur des SPPs dont les colonnes sont générées a priori. Ceci n'est généralement pas possible pour les problèmes de grande taille.

Dans la littérature, aucune méthode de génération de colonnes en nombres entiers avec décomposition n'a été proposée. Par conséquent, les sujets présentés dans cette thèse constituent les premiers travaux de recherche qui visent à exploiter les avantages du simplexe en nombres entiers avec décomposition pour résoudre efficacement des SPPs de grande taille. Les mé-

thodes développées dans le cadre de cette thèse se basent principalement sur les résultats théoriques présentés dans les articles [6, 7, 8].

2.3 Apprentissage machine

L'apprentissage machine (ML : machine learning) est un sous domaine de l'intelligence artificielle dédié au développement des systèmes intelligents permettant aux ordinateurs d'apprendre et d'évoluer eux-mêmes sans qu'ils soient programmés explicitement. Ces systèmes sont entraînés sur des données (e.g., des observations, des expériences passées) pour les analyser et déduire leurs structures statistiques et des règles qui constituent de nouvelles connaissances. Ces dernières sont utilisées par la suite pour une analyse rapide des données futures. Dans la littérature, on trouve quatre types de ML : apprentissage supervisé, non supervisé, semi-supervisé et apprentissage par renforcement. Dans cette section, nous présentons uniquement une brève revue de littérature sur l'apprentissage supervisé et l'apprentissage profond (sous-domaine de ML) utilisés dans le cadre de cette thèse, ainsi que certaines applications de ML dans le domaine de l'optimisation combinatoire.

2.3.1 Apprentissage supervisé

L'apprentissage machine est dit supervisé lorsqu'il est effectué sur un jeu de données $\mathcal{D}$ de type entrée-sortie $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$. Chaque entrée $\mathbf{x}_i \in \mathbf{X}$ est un vecteur de caractéristiques dans $\mathbb{R}^k$ où k est le nombre de caractéristiques. À chaque vecteur $\mathbf{x}_i \in \mathbf{X}$, une sortie $\mathbf{y}_i \in \mathbf{Y}$ est associée, où $\mathbf{Y}$ est l'ensemble des sorties possibles. Lorsque les $\mathbf{y}_i$ sont des valeurs continues, on parle alors d'un problème de *régression*, sinon, on parle d'un problème de *classification* dans le cas où les $\mathbf{y}_i$ sont des valeurs catégoriques. Ce problème est dit aussi de *classification multiclass*, lorsque l'ensemble $\mathbf{Y}$ contient plus que deux valeurs catégoriques possibles.

L'objectif de l'apprentissage supervisé est de trouver une fonction $f : \mathbf{X} \rightarrow \mathbf{Y}$ qui permet d'exprimer les sorties en fonction des entrées. Cette fonction est utilisée par la suite pour prédire la sortie $\mathbf{y}$ d'une nouvelle observation. Pour évaluer la qualité de la fonction f choisie, on utilise une fonction de perte qu'on cherche à minimiser :

$$\min \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(\mathbf{x}_i), \mathbf{y}_i) \quad (2.16)$$

La fonction de perte $\mathcal{L}$ peut être par exemple l'erreur quadratique pour les problèmes de régression ou la fonction d'entropie croisée pour les problèmes de classification.

Dans la littérature, plusieurs algorithmes d'apprentissage supervisé ont été proposés. Pour les

problèmes de régression, on trouve : régression linéaire, régression polynomiale, méthode du plus proche voisin, etc. Pour les problèmes de classification, on trouve : régression logistique, arbres de décisions, machines à vecteurs de support, etc.

2.3.2 Apprentissage profond

L'apprentissage profond [25] est un sous-domaine de ML qui a connu une grande progression durant les dernières années. Il regroupe plusieurs méthodes basées principalement sur les réseaux de neurones artificiels (inspirés du cerveau humain). Ces méthodes ont connu un grand succès grâce à leurs performances sur différents problèmes tels que la vision par ordinateur, la reconnaissance de la parole, le traitement du langage naturel et plus encore. Ce succès revient au progrès remarquable en puissance de calcul des machines, ainsi que la disponibilité de données massives. Les réseaux de neurones sont composés de trois parties principales :

- **Couche d'entrée** : Cette couche transmet les vecteurs de caractéristiques $\mathbf{x}_i$ au réseau de neurones. Elle est composée de plusieurs neurones, où chaque neurone est dédié à une caractéristique dans le vecteur d'entrée $\mathbf{x}_i$.
- **Couches cachées** : Appelées aussi couches profondes, ces couches permettent au réseau de neurones d'effectuer une série de transformations sur les entrées de chaque couche en les multipliant par certains poids $\mathbf{w}$ et en appliquant une fonction d'activation (e.g., ReLU, sigmoïde, ...). Il est à noter que la complexité du modèle est proportionnelle au nombre de couches cachées. Néanmoins, l'ajout de plus de couches peut permettre l'obtention d'une meilleure précision sur les données d'apprentissage.
- **Couche de sortie** : Cette couche rassemble les sorties de la dernière couche cachée du réseau et fournit la sortie finale. Pour les problèmes de classification, une fonction *softmax* est souvent utilisée pour transformer la sortie en probabilités selon les différentes classes dans $\mathbf{Y}$.

Une fonction de perte $\mathcal{L}$ est aussi utilisée en apprentissage profond pour évaluer la performance du réseau de neurones. Cette fonction mesure l'erreur de prédiction, c'est-à-dire la différence entre la valeur $f(\mathbf{x}_i)$ prédite par le modèle et le résultat observé $\mathbf{y}_i$. Les poids des neurones $\mathbf{w}$ sont ensuite ajustés à l'aide d'un processus appelé *rétropropagation du gradient*. L'objectif final est d'obtenir des poids $\mathbf{w}$ permettant de minimiser l'erreur. Néanmoins, même si l'erreur d'un modèle donné est petite sur les données d'apprentissage, le modèle peut ne pas être performant sur de nouvelles observations $(\mathbf{x}_i, \mathbf{y}_i)$ qui n'étaient pas utilisées durant l'entraînement du modèle. Des techniques de régularisation (e.g., *dropout* [26]) permettent de remédier à ce problème de *surapprentissage*.

Dans la littérature, nous trouvons plusieurs types de réseaux de neurones, entre autres, les réseaux convolutionnels et les réseaux récurrents. Le lecteur peut consulter le livre de Goodfellow et *al.* [25] pour plus de détails sur ces deux types de réseaux de neurones.

2.3.3 Apprentissage machine et optimisation combinatoire

Le succès remarquable des algorithmes d'apprentissage machine a rapidement suscité l'intérêt des chercheurs pour les utiliser dans le domaine de la recherche opérationnelle pour développer de nouvelles méthodes de résolution des problèmes combinatoires ou accélérer les méthodes existantes.

En 2014, Alvarez et *al.* [27] ont utilisé un modèle d'apprentissage supervisé dans le contexte du *Branch-and-Bound* pour apprendre une règle de branchement heuristique qui imite le branchement fort. Pour ce faire, les auteurs ont entraîné un modèle sur un ensemble de données collecté à partir des décisions de branchement prises à l'aide du branchement fort. Ce modèle représente la nouvelle règle de branchement heuristique qui permet de sélectionner la variable sur laquelle il faut brancher. Un travail similaire a été réalisé par Khalil et *al.* [28] qui ont utilisé un autre modèle pour imiter le branchement fort.

En 2018, Václavík et *al.* [29] ont utilisé un modèle de régression, dans le contexte du *Branch-and-Price*, pour estimer une borne supérieure, raisonnablement serrée, de la valeur optimale du SP. Le modèle est entraîné en temps réel sur les données des itérations passées pour prédire la prochaine borne supérieure. Cette borne est ensuite exploitée pour accélérer la résolution du SP. Les résultats numériques des tests, effectués sur des instances du problème d'horaires des infirmières et le problème de multiplexage temporel, montrent une réduction de 40% (en moyenne) du temps de résolution. Cependant, cette approche est seulement utilisable lorsque le SP peut profiter d'une borne supérieure, e.g., un programme linéaire en nombres entiers.

En 2019, Yaakoubi et *al.* [30] ont proposé un modèle d'apprentissage profond qui permet de prédire avec une certaine probabilité les connexions entre les vols du CPP. Ensuite, les auteurs se sont basés sur les résultats du modèle de prédiction pour développer des heuristiques qui construisent une solution initiale pour la méthode DCA [17] combinée avec la méthode de génération de colonnes. Les résultats des tests sur des instances de CPP montrent que la nouvelle solution initiale a permis une réduction moyenne entre 6.8% et 8.5% des coûts de solutions obtenues en utilisant une solution initiale standard.

Pour une revue sur les approches utilisant les algorithmes d'apprentissage machine pour la résolution des problèmes d'optimisation combinatoire, le lecteur peut consulter les revues de littérature de Bengio et *al.* [31] et Lodi et Zarpellon [32].

2.4 Domaines d'applications

Dans cette section, nous présentons une revue de littérature sur le VCSP et le CPP que nous utilisons dans les expérimentations numériques de nos méthodes.

2.4.1 Problème d'horaires de chauffeurs et d'autobus (VCSP)

C'est un problème de planification important qui se pose dans les entreprises de transport en commun. Ce problème vise à attribuer des véhicules aux trajets de bus et des chauffeurs à des tâches (segments de trajet). Traditionnellement, les deux étapes de planification ont été traitées séquentiellement où les horaires de véhicules sont déterminés avant les horaires d'équipage. Des méthodes heuristiques et exactes d'optimisation intégrée des deux problèmes ont été proposées pour traiter les différentes versions de ce problème (flotte homogène ou hétérogène, un ou plusieurs dépôts). Ce problème a été traité, entre autres, par Haase et *al.* [33] en utilisant la méthode de *Branch-and-Price*. Cet article donne une description détaillée du réseau sous-jacent au SP de la CG.

2.4.2 Problème de rotations d'équipages (CPP)

Le problème de rotations d'équipages dans le domaine aérien consiste à déterminer un ensemble de rotations permettant de couvrir tous les vols d'une période donnée où chaque vol est couvert par une seule rotation. Une rotation est une séquence de vols effectuée par un équipage en partant et revenant à la même base. Ce problème est habituellement résolu par la méthode *Branch-and-Price* (voir Desaulniers et *al.* [34]) où les rotations sont générées par les SPs qui sont modélisés comme des problèmes de plus court chemin avec contraintes de ressources [34]. Plus d'informations sur la structure exacte des SPs peuvent être trouvées dans [35].

CHAPITRE 3 ORGANISATION DU TRAVAIL

À travers la littérature, nous avons constaté que les méthodes primales ont connu une grande avancée suite à l'introduction de l'algorithme ISUD et ses variantes. Cet algorithme trouve fréquemment des solutions entières et non seulement il réduit les effets de la dégénérescence, il profite de celle-ci pour favoriser l'intégralité. Les résultats prometteurs obtenus par ISUD et ses variantes constituent notre principale motivation pour la réalisation des travaux de recherche présentés dans cette thèse. Le but ultime de cette dernière est de résoudre efficacement des problèmes industriels ayant une structure de SPP en produisant des solutions entières tout au cours de la résolution. La première étape consistait à introduire une première méthode primale de génération de colonnes en nombres entiers, nommée ICG, qui combine ISUD et GC pour résoudre les SPPs. Ensuite, nous proposons une méthode primale, nommée I²CG. C'est une adaptation d'ICG pour traiter les contraintes supplémentaires. Finalement, avec l'avènement récent des méthodes de prédiction performantes, nous avons vu l'opportunité de les exploiter pour concevoir une stratégie d'accélération d'I²CG.

Le chapitre 4 aborde la méthode ICG qui a pour objectif est de résoudre les problèmes dont le problème maître se formule comme un SPP. En effet, ICG consiste à résoudre le RMP en utilisant l'algorithme ISUD. Ensuite, ICG utilise une solution duale, correspondant à la meilleure solution entière trouvée, pour générer un grand nombre de colonnes favorisant l'intégralité des directions trouvées par le CP. Par conséquent, ICG trouve une suite de solutions entières menant à une solution optimale ou quasi-optimale.

Dans le chapitre 5, nous décrivons une adaptation de la méthode ICG pour résoudre efficacement les SPPSCs. Le choix de ce sujet est motivé par plusieurs facteurs. D'une part, le problème maître des problèmes industriels (e.g., CPP et VCSP) ne se formule pas nécessairement comme un SPP sans contraintes supplémentaires. D'autre part, ces dernières sont difficiles à gérer, dans un contexte primal, car elles peuvent briser la quasi-intégralité du SPP. En conséquence, une méthode primale (e.g., ISUD) peut être bloquée en un minimum local. Pour gérer ces contraintes, nous avons introduit une nouvelle formulation du RP et CP permettant de trouver des directions de descente entières qui améliorent, à chaque itération, la solution courante.

Dans le chapitre 6, nous présentons une amélioration de la méthode I²CG en utilisant des techniques d'apprentissage profond. En fait, nous exploitons un historique des solutions entières du CPP pour entraîner un réseau de neurones capable de produire les probabilités de connexions entre les vols. Ces dernières sont utilisées dans le SP pour favoriser la génération

des colonnes ayant une plus grande probabilité d'appartenir à des solutions optimales ou quasi-optimales. La nouvelle version d'I²CG trouve aussi une séquence de solutions entières avec des coûts décroissants jusqu'à atteindre une solution optimale ou quasi-optimale en réduisant les temps de résolution.

CHAPITRE 4 ARTICLE 1: INTEGRAL COLUMN GENERATION FOR THE SET PARTITIONING PROBLEM

Cet article a été publié dans la revue EURO Journal on Transportation and Logistics en juin 2019.

Référence : A. Tahir, G. Desaulniers et I. El Hallaoui, “Integral column generation for the set partitioning problem,” EURO Journal on Transportation and Logistics, p. 1–32, 2019.

4.1 Introduction

Many complex industrial problems are formulated as a set partitioning problem (SPP). These problems include, among others, the integrated vehicle and crew scheduling problem arising in public transit (VCSP, see Haase *et al.* [33]) and the airline crew pairing problem (CPP, see Desaulniers *et al.* [34]) which are often solved by branch-and-price (see Barnhart *et al.* [10]). The SPP is a combinatorial optimization problem that can be defined as follows. Let $T = \{1, \dots, m\}$ be a set of tasks to accomplish exactly once (e.g., flights to operate, customers to visit once each, etc.). Let $N = \{1, \dots, n\}$ be a set of feasible task subsets (e.g., defined by crew schedules, vehicle routes, etc.). For each task subset $j \in N$, let c_j be the least cost to accomplish these tasks and let a_{ij} , $i \in T$, be a binary parameter indicating if task i is in subset j or not. The SPP consists of selecting subsets in N such that each task in T belongs to exactly one of the selected subset and the sum of the costs of these subsets is minimized. It can be formulated as the following integer program:

$$\min_x \quad \sum_{j \in N} c_j x_j \quad (4.1)$$

$$\text{s.t.:} \quad \sum_{j \in N} a_{ij} x_j = 1, \quad \forall i \in T \quad (4.2)$$

$$x_j \in \{0, 1\}, \quad \forall j \in N, \quad (4.3)$$

where x_j , $j \in N$, is a binary variable equal to 1 if subset j is selected and 0 otherwise. The objective function (4.1) aims at minimizing the total cost. The set partitioning constraints (4.2) ensure that each task is included in a single selected subset. Finally, binary requirements on the x_j variables are expressed by (4.3). Note that additional (non-set-partitioning) constraints can be considered to model other aspects of the problem at hand such as vehicle availability in vehicle scheduling problems. In this work, we focus on the pure SPP, i.e., without additional constraints.

In the following, we denote by H the convex hull of the set of the feasible solutions of (4.1)–(4.3) and by R the feasible region of the linear relaxation of (4.1)–(4.3). Notice that H and R are convex polytopes if they are non-empty.

Since 1969, researchers have been attracted by the SPP because its polytope R possesses the *quasi-integrality* property. This property, which distinguishes the SPP from most of the other problems, stipulates that every edge of H is also an edge of R . This implies that every integer solution corresponding to an extreme point of H is also an extreme point of R and, for every pair of such integer points, there exists on the boundary of R a path linking them which is composed of edges linking only integer extreme points. The study of this property has led to the development of several interesting solution algorithms (see Balas and Padberg [1], Thompson [2], and Saxena [3]), including the Integral Simplex Using Decomposition (ISUD) algorithm of Zaghroui *et al.* [6]. These algorithms will be reviewed in the next section.

This paper continues these works on the SPP. Indeed, we exploit existing theoretical results to propose a new Integral Column Generation (ICG) algorithm. Starting from a possibly poor-quality solution that might even be infeasible, this new primal method finds a sequence of improving feasible integer solutions. At the opposite of the branch-and-price algorithm which only uses dual information to generate new variables, the ICG algorithm exploits both primal and dual information to do so. On the one hand, the current dual solution is used to find new columns (task subsets) that can potentially be part of an improved integer solution. On the other hand, the primal information corresponding to the current integer solution is exploited to find descent directions yielding better integer solutions. To increase the number of descent directions found at each iteration of the ICG algorithm, we have chosen to generate a very large number of columns each time that the column generation subproblem is solved. In a traditional column generation algorithm, this strategy is avoided as it typically increases substantially the time to re-optimize the restricted master problem (RMP) and, therefore, the total computational time. However, in the ICG algorithm, it is rather beneficial as it increases the probability of finding several improved integer solutions. Furthermore, it does not bother the solution of the RMP because this one is decomposed and the density of the constraint coefficient matrix of the original problem is reduced as explained in Section 4.3.

In practice, large SPP instances are often solved using a column generation (CG) heuristic; for example, a diving heuristic (DH) or a restricted master heuristic (RMH) as briefly described in Section 4.5. Such heuristics usually do not find many integer solutions (and may even fail to do so). Moreover, when some are found, the good ones often appear only towards the end of the solution process, leaving no opportunity to the planner to stop prematurely the solution process when he/she is satisfied with the quality of the incumbent solution. In this

paper, we aim at solving large SPP instances while offering this opportunity to the planner. In this context where optimality is not necessarily sought, we have designed and implemented a heuristic version of the ICG algorithm although it could have been exact. We have tested this version on instances of the VCSP and of the CPP, with up to 2000 and 1740 constraints in the master problem, respectively. Our computational results show the effectiveness of the proposed algorithm compared to two popular CG heuristics, namely, DH and RMH. Indeed, we succeed to compute a better quality solution (often optimal or near-optimal) in much less computational time for most instances. This is achieved by performing a small number of column generation iterations (at most 12 for the VCSP and 23 for the CPP) and finding a large number of integer solutions throughout the solution process (around 10 solutions per column generation iteration on average).

This paper is structured as follows. In the next section, we review the literature on the SPP and on the primal and dual-fractional algorithms that are the most commonly used for solving this problem. In Section 4.3, we present the ISUD algorithm and related concepts which are at the basis of the ICG algorithm. Section 4.4 describes this new algorithm. Section 4.5 reports the results of our computational experiments. Finally, conclusions are drawn in Section 4.6.

4.2 Literature review

In the literature on the SPP, there are two main types of solution algorithms, namely, dual-fractional and primal algorithms. Primal algorithms move from one feasible (integer) solution to another, whereas dual-fractional algorithms allow infeasible (non-integer) solutions. In this review, we focus in Section 4.2.1 on a single family of dual-fractional algorithms, the branch-and-price (BP) algorithms which can handle SPP instances with a very large number of variables, and in Section 4.2.2 on two types of primal algorithms, namely, the integral simplex algorithms and the ISUD algorithm.

4.2.1 Branch-and-price algorithms

Branch-and-price (see Barnhart *et al.* [10]) is a popular and efficient solution method for the SPP. It embeds column generation in a branch-and-bound framework, where a linear relaxation of SPP is solved using CG at each node of the search tree. CG is designed to solve linear programs containing a very large number of variables that are often obtained through Dantzig-Wolfe decomposition (Dantzig and Wolfe [11]). In this context, the linear program is called the *master problem* (MP) and the CG algorithm solves at each iteration the

MP restricted to a subset of the variables, called the *RMP*, and one or several *subproblems*. In our case, the MP corresponds to the linear relaxation of (4.1)–(4.3) and the subproblem is often modeled as a shortest path problem with resource constraints (SPPRC, see [36]). Solving the RMP provides a primal solution but also a dual solution denoted $\alpha \in \mathbb{R}^m$. The subproblem aims at finding new columns associated with variables x_j of negative reduced cost $\bar{c}_j = c_j - \sum_{i \in T} \alpha_i a_{ij}$ that are added to the RMP before starting a new iteration. If no such columns can be generated, then the CG algorithm stops and the optimal solution of the current RMP is also optimal for the MP (setting all unknown variables to 0).

It is well known that the standard CG algorithm may be subject to several convergence issues (see Vanderbeck [16]) which transfer to the branch-and-price algorithm. In the literature, several dual variable stabilization techniques have been introduced to improve convergence (see, e.g., [15, 16]). In 2005, Elhallaoui *et al.* [17] have proposed the dynamic constraint aggregation (DCA) algorithm for solving the SPP. This algorithm consists of reducing the number of set partitioning constraints by aggregating some of them as needed. The aggregation is revised during the solution process to ensure the exactness of the DCA algorithm. The authors report computational results obtained on randomly generated VCSP instances which show that DCA clearly outperforms standard CG for solving the linear relaxation of the SPP: computational time reductions of up to 80% were achieved. An improved variant of the DCA algorithm called the multi-phase DCA (MPDCA) algorithm was also developed by Elhallaoui *et al.* [18]. This variant allows a further reduction of degeneracy and of the number of fractional-valued variables in the computed MP solutions. Consequently, it often finds integer solutions. More recently, to alleviate the drawbacks of the standard CG algorithm, Bouarab *et al.* [20] proposed three new decompositions that integrate the improved primal simplex (IPS) algorithm of Elhallaoui *et al.* [19] and either CG or DCA. The best decomposition combines IPS and DCA and reduces the number of non-zero elements in the constraint coefficient matrix. This opens up the possibility to consider more columns in the RMP and, thus, to stabilize the dual variables. The computational results reported by Bouarab *et al.* [20] show that their IPS/DCA algorithm can be eight times faster than the standard CG algorithm on the instances tested by Elhallaoui *et al.* [17, 18]. These results also show that the application of a clever decomposition in a CG framework is a promising research avenue that is worth exploring to overcome the convergence issues associated with standard CG.

4.2.2 Primal algorithms

Here, we review first the integral simplex algorithms without decomposition before discussing the ISUD algorithm.

Without decomposition

In 1969, Trubin [21] proved that the SPP polytope has the quasi-integrality property: every edge of H is an edge of R . This property implies that, for every pair of feasible solutions, there exists a path between them that is composed of edges of R , visits only integer solutions at the edge extremities, and the sequence of the costs of the visited solutions is decreasing (see Balas and Padberg [22]). These authors also showed that an optimal solution can be reached in at most m pivots from any initial feasible solution. This result is, however, impractical because it requires the knowledge of this optimal solution. Based on these results, Thompson [2] introduced in 2002 the integral simplex algorithm, which proceeds in two phases. The first phase (local method) consists of performing, as long as possible, non-degenerate pivots to reach a local optimum. All these pivots are performed on a coefficient (that of the entering variable in the row associated with the leaving variable) equal to 1. The second phase (global method) builds a tree, where each node corresponds to a subproblem that is solved using the local method. An optimal solution to SPP is given by the best solution found in the nodes explored in the tree. Soon after, Saxena [3] proposed an improvement to Thompson's algorithm that allows pivoting on a -1 coefficient and relies on anti-cycling rules. The great advantage of these integral simplex algorithms is that they always have a feasible integer solution at hand. However, these algorithms suffer from degeneracy and might struggle to find a non-degenerate pivot.

Rönnberg and Larsson [4, 5] have developed combined CG/integral simplex algorithms, called all-integer column generation algorithms. They first adapted the necessary and sufficient condition on the columns yielding non-degenerate pivots proposed by Balas and Padberg [22] and derived from it a condition to identify columns yielding degenerate pivots. Depending on the type of columns to be generated, one of these conditions is integrated in the column generation subproblem to restrict the set of columns that can be generated. The local method of the integral simplex algorithm iterates between performing non-degenerate pivots in the RMP, generating columns yielding non-degenerate pivots, performing degenerate pivots in the RMP, and generating columns yielding degenerate pivots, until reaching a stopping criterion. Then, the global method of the integral simplex algorithm which explores a search tree is applied to complete the solution process. The authors have described academic examples to illustrate the unfolding of this algorithm but no extensive computational experiments were

conducted. Tests on small-sized instances of the generalized assignment problem show that degeneracy remains an issue.

Note that these algorithms generate sequences of improving integer solutions. To move from one solution to the next, they perform a (possibly empty) sequence of degenerate pivots followed by a non-degenerate pivot. The selection of the degenerate pivots in each sequence is somewhat arbitrary and, thus, may yield long sequences and waste substantial computational time.

With decomposition

In 2011, Elhallaoui *et al.* [19] introduced the improved primal simplex (IPS) algorithm for solving efficiently linear programs subject to high degeneracy. The success of this algorithm has motivated an attempt to adapt it to an integer program, namely, the SPP. In 2014, Zaghroui *et al.* [6] realized this IPS adaptation by introducing a new algorithm called the ISUD algorithm. As in the IPS algorithm, the ISUD algorithm relies on a decomposition concept specialized for degenerate problems. Indeed, it decomposes the SPP in a reduced problem (RP) and a complementary problem (CP) that searches for descent directions that can improve the current integer solution. The definitions of RP and CP as well as the ISUD algorithm will be given in Section 4.3. Compared to the algorithms described in Section 4.2.2, the ISUD algorithm finds at each iteration a combination of columns to pivot into the basis. This combination ensures a decrease in the objective function and is minimal in the sense that, if one of the columns it contains is not pivoted into the basis, then no improvement can be achieved. The computational experiments performed by Zaghroui *et al.* [6] on instances of the VCSP and of the CPP involving up to 1,600 constraints and 500,000 variables showed the effectiveness of the ISUD algorithm to find optimal or near-optimal solutions much more rapidly than the CPLEX mixed-integer programming (MIP) solver for the 90 tested instances. Note, however, that, in these instances, all columns are known a priori.

To improve the ISUD algorithm, further research has been realized by Rosat *et al.* [23, 24] and Zaghroui *et al.* [7]. These works focused on a weakness of the ISUD algorithm, namely, solving the CP may sometimes require some kind of branching to ensure finding a descent direction leading to an integer solution, called hereafter an *integer direction*. Rosat *et al.* [23] have proposed cuts to avoid branching. These cuts are, however, costly to separate and several of them might be needed to find an integer direction. Alternatively, Rosat *et al.* [24] studied the role of the normalization constraint added to the CP to bound it. The left-hand side of this constraint is defined as a weighted sum of variables. The authors showed the existence of a weight vector that enables finding only integer directions until reaching optimality.

Unfortunately, no procedure for building this vector is proposed. Finally, Zaghroui *et al.* [7] developed a variant of the ISUD algorithm called the zooming algorithm. When it is not possible to find an integer direction without branching, the CP provides a non-integer descent direction that can be used to define a neighborhood around the current integer solution. This neighborhood is of small size and can, thus, be explored efficiently using a MIP solver to find an integer direction.

From this literature review, we observe that the best primal algorithm for the SPP, namely, the ISUD algorithm, has not been adapted to the CG context. Consequently, this paper constitutes the first attempt at combining CG and ISUD to compute efficiently sequences of improving integer solutions for large-scale SPP instances.

4.3 The ISUD algorithm

In this section, we describe the ISUD algorithm of Zaghroui *et al.* [6] which is needed to understand the proposed ICG algorithm and apply it to a small example. In the following, we use bold characters to denote vectors and matrices.

4.3.1 Description of the ISUD algorithm

As mentioned above, the ISUD algorithm is based on a decomposition of the problem into a RP and a CP. Therefore, before describing this algorithm, we introduce the RP and the CP whose definitions rely on a compatibility criterion which is presented first.

Let $\mathbf{x}$ be a feasible solution to SPP (4.1)–(4.3) and $S = \{j \mid x_j = 1\}$ the set of the indices of the task subsets selected in this solution. Let $\mathbf{A} = (a_{ij})_{i \in T, j \in N}$ be the constraint coefficient matrix and denote by $\mathbf{A}_j$ its column associated with variable x_j , $j \in N$. By breach of terminology, we say that S is the solution $\mathbf{x}$ and that the columns $\mathbf{A}_j$, $j \in S$, are in the solution $\mathbf{x}$ or in S . The other columns $\mathbf{A}_j$, $j \in N \setminus S$, are either *compatible* or *incompatible* with the columns in S according to the following definition.

Definition 1. *An arbitrary column in $\{0, 1\}^m$ (not necessarily a column of $\mathbf{A}$) is said to be compatible with the columns in S (or, simply, compatible with S) if it can be written as a linear combination of these columns.*

Let $\mathbf{A}_U = (\mathbf{A}_j)_{j \in U}$ be a submatrix of $\mathbf{A}$. We denote by $\mathbf{A}_U^1$ the submatrix of $\mathbf{A}_U$ composed of its first $|U|$ linearly independent rows and all its columns and by $\mathbf{A}_U^2$ the submatrix of $\mathbf{A}_U$ containing the remaining rows. Furthermore, let C_S be the index set of the columns $\mathbf{A}_j$, $j \in N$, that are compatible with S and let $I_S = N \setminus C_S$ be the index set of the incompatible

columns. Observe that $S \subseteq C_S$ and that $\mathbf{A}_{C_S}^1$ contains $|S|$ rows, namely, one for the first task covered by each column $\mathbf{A}_j$, $j \in S$. As mentioned in the previous section, the SPP is decomposed in two problems: a RP built from the compatible columns and a CP containing the incompatible ones.

The RP is given by:

$$\min_{\mathbf{x}_{C_S}} \quad \mathbf{c}_{C_S}^\top \mathbf{x}_{C_S} \quad (4.4)$$

$$\text{s.t.:} \quad \mathbf{A}_{C_S}^1 \mathbf{x}_{C_S} = \mathbf{e} \quad (4.5)$$

$$\mathbf{x}_{C_S} \in \{0, 1\}^{|C_S|} \quad (4.6)$$

where $\mathbf{x}_{C_S}$ is the subvector of the variables associated with the columns compatible with S , $\mathbf{c}_{C_S}$ is the subvector of the cost coefficients associated with these variables, and $\mathbf{e} \in \{1\}^{|S|}$ is a vector of ones. In RP, the current solution S is associated with a basis corresponding to the identity matrix. Consequently, pivoting into the basis any compatible column that has a negative reduced cost results in an improved integer solution S' (the entering column replaces two or more columns in the solution). This first pivot is, thus, non-degenerate. One can then redefine RP with respect to the new solution S' and search for a new negative reduced cost column in $C_{S'}$ that is compatible with S' . This process can be repeated until no such column is found.

The incompatible columns $\mathbf{A}_j$, $j \in I_S$, are used in the CP to find an integer descent direction. This direction is obtained by finding a linear combination of the incompatible columns that is compatible with S and composed of disjoint columns according to the following definition.

Definition 2. Let $U \subseteq N$ be a subset of column indices. The columns $\mathbf{A}_j$, $j \in U$, are said to be disjoint if and only if $\mathbf{A}_{j_1}^\top \mathbf{A}_{j_2} = 0$ for all $j_1, j_2 \in U$ such that $j_1 \neq j_2$.

Let v_j , $j \in I_S$, be the weight variables defining the linear combination of the incompatible columns; λ_l , $l \in S$, the weight variables defining the linear combination of the columns in the solution S ; and w_j , $j \in I_S$, nonnegative weights used in the normalization constraint. Furthermore, denote by $F_S = \{(j_1, j_2) \in I_S \times I_S \mid \mathbf{A}_{j_1}^\top \mathbf{A}_{j_2} \neq 0\}$ the set of index pairs of non-disjoint incompatible columns. The CP can be formulated as follows:

$$z^{CP} = \min_{\mathbf{v}, \boldsymbol{\lambda}} \sum_{j \in I_S} c_j v_j - \sum_{l \in S} c_l \lambda_l \quad (4.7)$$

$$\text{s.t.:} \quad \sum_{j \in I_S} v_j \mathbf{A}_j - \sum_{l \in S} \lambda_l \mathbf{A}_l = 0 \quad (4.8)$$

$$\sum_{j \in I_S} w_j v_j = 1 \quad (4.9)$$

$$v_j \geq 0, \quad \forall j \in I_S \quad (4.10)$$

$$v_{j_1} v_{j_2} = 0, \quad \forall (j_1, j_2) \in F_S. \quad (4.11)$$

The objective function (4.7) searches for a direction that will incur a cost decrease. Constraint (4.8) ensures that the linear combination of the incompatible columns is compatible with S . The normalization constraint (4.9) simply bounds the feasible region. Nonnegativity constraints on the v_j variables are enforced through (4.10). Given these nonnegativity requirements and the facts that the elements of $\mathbf{A}$ are binary and the columns in S are disjoint, the λ_l variables are also nonnegative. Finally, the disjunctive constraints (4.11) impose the selection of disjoint columns in the linear combination of the incompatible variables, ensuring an integer direction.

If $z^{CP} < 0$, then an integer descent direction $\mathbf{d} = (d_j)_{j \in N}$ is identified by the positive-valued v_j and λ_l variables as follows:

$$d_j = \begin{cases} 1 & \text{if } j \in I_S \text{ and } v_j > 0 \\ -1 & \text{if } j \in S \text{ and } \lambda_j > 0 \\ 0 & \text{otherwise,} \end{cases} \quad \forall j \in N. \quad (4.12)$$

Fixing the step length to 1, an improved integer solution is obtained by replacing the columns $\mathbf{A}_l$, $l \in S$, with $\lambda_l > 0$ by the incompatible columns $\mathbf{A}_j$, $j \in I_S$, with $v_j > 0$. This corresponds to entering the latter columns into the basis, while removing the former from it. On the other hand, if $z^{CP} \geq 0$, then no integer descent direction can be found and solution S is, therefore, optimal.

In practice, constraints (4.11) are relaxed from the initial CP, yielding the *relaxed CP* which can then be solved by a linear programming solver. As empirically shown by Zaghrouti *et al.* [6], the selected columns in the linear combination of incompatible columns are often disjoint and the computed solution is thus feasible with respect to the relaxed constraints (4.11). When this is not the case, branching can be performed to enforce these constraints. For example, for each column $\mathbf{A}_j$, $j \in I_S$, with $v_j > 0$ in the solution of the relaxed CP, one can

create a child node that imposes $v_j = 0$.

The main steps of the ISUD algorithm described above are summarized in Algorithm 1. The performance of this algorithm mainly depends on the effectiveness at solving the CP in Step 3. Below, we discuss different avenues that have been studied to ease its solution.

Algorithm 1: ISUD algorithm

- 1: Start from an initial solution S
 - 2: Improve the current solution S through the RP
 - 3: Solve the CP
 - 4: **if** $z^{CP} < 0$ **then**
 - 5: Update solution S according to the integer direction found
 - 6: Go to Step 2
 - 7: **return** the current solution S
-

Observe that the density of the constraint coefficient matrix in the relaxed CP is very similar to that of the original SPP. To reduce this density, one can reformulate the CP by performing a variable substitution and eliminating a number of constraints. Several reformulations are possible as discussed in Appendix A.1. We have chosen the one based on matrix $\mathbf{M}_2$ that was proposed by Bouarab *et al.* [20]. Another option to reduce the matrix density is to apply the multi-phase acceleration strategy devised by Zaghrouti *et al.* [6] and presented in Appendix A.2. It consists of solving a sequence of CPs by considering in phase k a CP that contains only the incompatible columns which have an incompatibility degree less than or equal to k . When matrix $\mathbf{M}_2$ is used to define it, the *degree of incompatibility* δ_j of a column $\mathbf{A}_j$, $j \in I_s$, indicates the minimum number of times that task subsets in the current solution S must be divided to ensure that $\mathbf{A}_j$ becomes compatible with the resulting task subsets. For example, if the current solution S contains the two task subsets $\{1, 2, 3\}$ and $\{4, 5\}$, then the degree of incompatibility of a column $\mathbf{A}_j$ covering the task subset $\{1, 2, 5\}$ is equal to 2 because to make it compatible both subsets $\{1, 2, 3\}$ and $\{4, 5\}$ need to be split up. It was shown by Bouarab *et al.* [20] that, in phase k , any column in the coefficient matrix of the CP has at most $k + 1$ non-zero coefficients.

In a CG context like in the proposed ICG algorithm, a complete dual solution $\boldsymbol{\alpha} = (\alpha_i)_{i \in T} \in \mathbb{R}^m$ of the MP (i.e., the linear relaxation of (4.1)–(4.3)) is required to generate negative reduced cost columns. One drawback of using a reformulation of the CP is that no such dual solution is available. Appendix A.3 discusses how one can be retrieved from the dual solution of the relaxed CP. The proposed technique ensures that the reduced cost of every column in the current solution S with respect to this dual solution $\boldsymbol{\alpha}$ is equal to zero. In this case, we

say that the dual solution *corresponds to solution S*.

Finally, note that, for our tests, we have chosen to set $w_j = \delta_j$ for all $j \in I_S$. Rosat *et al.* [24] showed that using these weights in the normalization constraint (4.9) favors finding integer directions in the relaxed CP.

4.3.2 A small example

To illustrate the ISUD algorithm, we apply it to a small example, namely, to the following set partitioning model:

$$\begin{array}{llllllllllll}
 \min_{\mathbf{x}} & 5x_1 + 5x_2 + 5x_3 + 2x_4 + 9x_5 + 6x_6 + 6x_7 + 5x_8 + 5x_9 + 5x_{10} + 9x_{11} \\
 & x_1 & & + & x_5 + x_6 & & + & x_8 & & + & x_{10} & & = & 1 \\
 & x_1 & & + & x_5 + x_6 & & + & x_8 & & + & x_{10} & & = & 1 \\
 & & x_2 & & + & x_5 + x_6 & & + & x_8 + x_9 & & & & = & 1 \\
 & & x_2 & & + & x_5 & & + & x_7 + x_8 & & + & x_{10} & & = & 1 \\
 & & & x_3 & & & & + & x_7 + x_8 + x_9 & & & + & x_{11} & = & 1 \\
 & & & x_3 & & & & + & x_7 & & + & x_9 + x_{10} + x_{11} & = & 1 \\
 & & & & x_4 & & & & & & & + & x_{11} & = & 1 \\
 & & & & x_4 & & & & & & & + & x_{11} & = & 1 \\
 & & & & & & & & & & & & & \mathbf{x} \in \{0, 1\}^{11}
 \end{array}$$

To solve this example using the ISUD algorithm, we proceed through the following steps.

1. Let $x_1 = x_2 = x_3 = x_4 = 1, x_5 = \dots = x_{11} = 0$ be an initial solution with cost 17. Therefore, $S = \{1, 2, 3, 4\}$, $C_S = S \cup \{5, 11\}$ (because $\mathbf{A}_5 = \mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{A}_{11} = \mathbf{A}_3 + \mathbf{A}_4$) and $I_S = \{6, 7, 8, 9, 10\}$.
2. The corresponding RP is built as:

$$\begin{array}{llllllll}
 \min_{\mathbf{x}_{C_S}} & 5x_1 + 5x_2 + 5x_3 + 2x_4 + 9x_5 + 9x_{11} \\
 & x_1 & & & + & x_5 & & = & 1 \\
 & & x_2 & & + & x_5 & & = & 1 \\
 & & & x_3 & & & + & x_{11} & = & 1 \\
 & & & & x_4 & & + & x_{11} & = & 1 \\
 & & & & & & & \mathbf{x}_{C_S} \in \{0, 1\}^6,
 \end{array}$$

which contains a representative constraint for each task subset in the current solution S and only the variables that are compatible with S .

3. Solving this RP yields an improved solution $S = \{3, 4, 5\}$ (with cost 16), from which we compute the index sets $C_S = S \cup \{11\}$ and $I_S = \{1, 2, 6, 7, 8, 9, 10\}$.

4. The ensuing relaxed CP is as follows:

$$\begin{aligned}
z^{CP} = \min_{\mathbf{v}, \boldsymbol{\lambda}} \quad & 5v_1 + 5v_2 + 6v_6 + 6v_7 + 5v_8 + 5v_9 + 5v_{10} - 5\lambda_3 - 2\lambda_4 - 9\lambda_5 \\
& v_1 + v_6 + v_8 + v_{10} - \lambda_5 = 0 \\
& v_1 + v_6 + v_8 + v_{10} - \lambda_5 = 0 \\
& v_2 + v_6 + v_8 + v_9 - \lambda_5 = 0 \\
& v_2 + v_7 + v_8 + v_{10} - \lambda_5 = 0 \\
& \quad + v_7 + v_8 + v_9 - \lambda_3 = 0 \\
& \quad + v_7 + v_9 + v_{10} - \lambda_3 = 0 \\
& \quad \quad - \lambda_4 = 0 \\
& \quad \quad - \lambda_4 = 0 \\
& v_1 + v_2 + v_6 + v_7 + v_8 + v_9 + 2v_{10} = 1 \\
& \mathbf{v} \geq 0.
\end{aligned}$$

Observe that the coefficient of v_{10} in the normalization constraint is 2 because the degree of incompatibility of $\mathbf{A}_{10}$ is $\delta_{10} = 2$, i.e., two task subsets of the current solution S need to be divided to make v_{10} compatible (see Zaghrouti *et al.* [6]). This coefficient is equal to 1 for all the other incompatible variables.

5. Solving this CP gives the optimal solution $\lambda_3 = \lambda_5 = 2/3$, $v_8 = v_9 = v_{10} = 1/3$, $v_j = 0$, $j \in \{1, 2, 6, 7\}$, with a cost of $z^{CP} = -13/3$. Because $(8, 9) \in F_S$, i.e., the columns $\mathbf{A}_8$ and $\mathbf{A}_9$ are not disjoint, and $v_8 v_9 \neq 0$, this solution does not yield an integer direction and branching is required to solve the non-relaxed CP.
6. Imposing $v_8 = 0$ as a first branching decision and re-optimizing the relaxed CP gives the optimal solution $\lambda_3 = \lambda_5 = 1/2$, $v_6 = v_7 = 1/2$, $v_j = 0$, $j \in \{1, 2, 8, 9, 10\}$, with a cost of $z^{CP} = -1$, which induces an integer descent direction $\mathbf{d} = (0, 0, 0, -1, -1, 1, 1, 0, 0, 0, 0)^\top$. This direction corresponds to replacing the columns $\mathbf{A}_3$ and $\mathbf{A}_5$ in the current solution by the columns $\mathbf{A}_6$ and $\mathbf{A}_7$, yielding a cost reduction of 2.
7. Updating the solution yields $S = \{4, 6, 7\}$, $C_S = \emptyset$, and $I_S = \{1, 2, 3, 5, 8, 9, 10, 11\}$.
8. Because $C_S = \emptyset$, the corresponding RP is trivial and the current solution cannot be improved by considering only the compatible variables.
9. The relaxed CP associated with S is built (not shown here) and solved. Its optimal solution is given by $\lambda_6 = \lambda_7 = 2/3$, $v_8 = v_9 = v_{10} = 1/3$, $v_j = 0$, $j \in \{1, 2, 3, 5, 11\}$, with a cost of $z^{CP} = -3$. However, this solution does not correspond to an integer direction and imposing separately the branching decisions $v_8 = 0$, $v_9 = 0$, and $v_{10} = 0$ indicates that there are no integer descent directions (i.e., $z^{CP} \geq 0$ for all three decisions). Consequently, the algorithm stops with S as an optimal solution.

4.4 Methodology

In this section, we start by describing the ICG algorithm before presenting an acceleration strategy. In the rest of the text, the words *variable* and *column* are used interchangeably.

4.4.1 The ICG algorithm

The ICG algorithm is based on the three-level decomposition illustrated in Figure 4.1. The first two levels correspond to the RMP in a column generation algorithm except that this RMP is an integer linear program, not a continuous linear program. The RMP is decomposed into a RP and a CP, and solved using the ISUD algorithm. The third level of the decomposition contains the column generation subproblem which generates negative reduced cost columns that are added to the pool of columns available for the RMP.

In our tests, the subproblem can be separated into several subproblems, which are SPPRCs defined on application-specific networks. In a SPPRC, resource constraints are used to enforce path feasibility rules such as a maximum flying time per day for an aircraft pilot or a minimum working time before a break for a bus driver. Typically, there is one constrained resource for each rule handled by such constraints. The subproblems are solved by a labeling algorithm. This algorithm represents a partial path from the source node of the network to any node by a label, i.e., a multi-dimensional vector with one component providing the (reduced) cost of the partial path and one component for each resource indicating the amount of this resource consumed along the path. It proceeds as follows (see [36] for details). Starting from an initial label at the source node, it extends labels using resource extension functions in the network to generate partial paths. After each extension, the resulting path is checked for feasibility, i.e., its label resource components must fall within predefined resource windows. Labels corresponding to infeasible paths are discarded. Labels can also be eliminated if they are dominated by other labels, i.e., if it can be proven that they cannot yield a Pareto-optimal path. The label extension process continues until all labels have been extended or discarded. Labels at the sink node(s) that have a negative cost can then be considered by the RMP as discussed below.

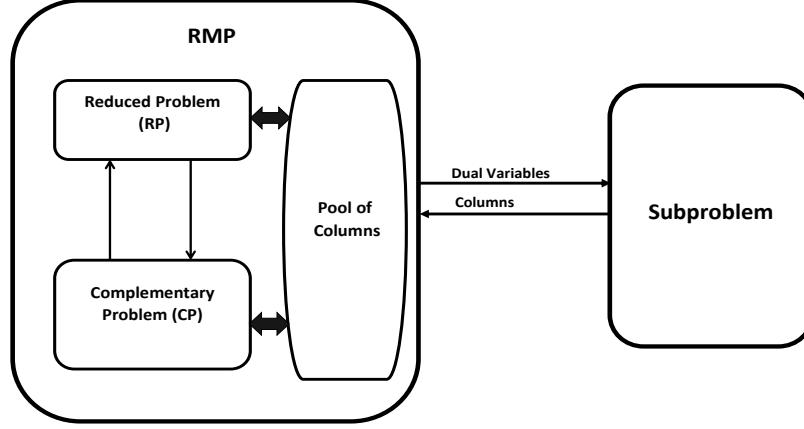


Figure 4.1 The three-level decomposition of the ICG algorithm

A pseudo-code of the ICG algorithm is given in Algorithm 2. It starts by computing an initial primal solution and an initial dual solution. In Section 4.5, we will describe how these solutions are computed for our test problems. It then initializes an iteration counter r , a current upper bound ℓ on the degree of incompatibility, and the number *consFail* of consecutive iterations in which the cost improvement (compared to the previous iteration) is not considered sufficient. As in the ISUD algorithm, the ICG algorithm applies a multi-phase strategy when generating the columns. In fact, the subproblem includes an additional resource constraint which imposes that the degree of incompatibility of the feasible columns be less than or equal to ℓ . In the associated resource extension functions, the consumption of this resource is increased by one each time that a new task $j \in T$ is covered, the last task previously covered by the extended path is $i \in T$ and i is not followed by j in the current solution S . Note that this multi-phase strategy is independent of the one used in the ISUD algorithm when solving the RMP (Steps 12 and 17).

The main loop (Steps 3–21) starts by solving the subproblem denoted $SP(\alpha^r, \ell)$ taking into account the current dual solution α^r and the current limit on the degree of incompatibility ℓ . The negative reduced cost columns generated by the subproblem, if any, are stored in set N' . If no columns are generated, ℓ is increased by one and the subproblem is solved again. This process is repeated until finding negative reduced cost columns or until ℓ exceeds a predefined value *maxDegree*. In the latter case, the algorithm stops and outputs (in Step 9) the current solution x^r as the best solution found. Note that, in practice, we try to generate a large number of columns at each iteration. Given that a multi-phase strategy is used in ISUD when solving the RMP, this large number of columns does not hinder the solution process.

When columns are found ($N' \neq \emptyset$), they are added to the pool of columns N available to the

Algorithm 2: ICG algorithm

```

1: Find an initial primal solution  $\mathbf{x}^0$  and an initial dual solution  $\boldsymbol{\alpha}^0$ 
2:  $r \leftarrow 0$ ;  $\ell \leftarrow 1$ ;  $consFail \leftarrow 0$ 
3: while  $consFail \leq maxConsFail$  do
4:   repeat
5:      $N' \leftarrow \text{Solve } SP(\boldsymbol{\alpha}^r, \ell)$ 
6:     if  $N' = \emptyset$  then
7:        $\ell \leftarrow \ell + 1$ 
8:       if  $\ell > maxDegree$  then
9:         return  $\mathbf{x}^r$ 
10:    until  $N' \neq \emptyset$ 
11:     $N \leftarrow N \cup N'$ ;  $r \leftarrow r + 1$ 
12:     $(z^r, \mathbf{x}^r, \boldsymbol{\alpha}^r) \leftarrow \text{Solve } RMP(\mathbf{x}^r)$  using ISUD
13:    if  $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$  then
14:       $(z^r, \bar{\mathbf{x}}^r) \leftarrow \text{Solve } MIP(\mathbf{x}^r)$ 
15:      if  $\bar{\mathbf{x}}^r \neq \mathbf{x}^r$  then
16:         $\mathbf{x}^r \leftarrow \bar{\mathbf{x}}^r$ 
17:       $(z^r, \mathbf{x}^r, \boldsymbol{\alpha}^r) \leftarrow \text{Solve } RMP(\mathbf{x}^r)$  using ISUD
18:      if  $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$  then
19:         $consFail \leftarrow consFail + 1$ ;  $\ell \leftarrow \ell + 1$ 
20:      else
21:         $consFail \leftarrow 0$ 
22: return  $\mathbf{x}^r$ 

```

RMP. The RMP is then solved by the ISUD algorithm which returns the current solution $\mathbf{x}^r$, its cost z^r and a dual solution $\boldsymbol{\alpha}^r$ corresponding to this current solution. This dual solution is computed as described in Section A.3. In the ISUD algorithm, a multi-phase strategy is applied and a relaxed CP (without constraints (A.5)) is always solved, i.e., branching is never performed to obtain an integer solution. In practice, the ISUD algorithm often terminates because the direction found when solving the last relaxed CP is not integer.

If the cost of the solution returned by ISUD does not yield a sufficiently large improvement compared to the cost achieved at the previous iteration, i.e., if $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImp$, where $minImp$ is a predefined parameter value, then a small-sized mixed integer program, denoted $MIP(\mathbf{x}^r)$, is solved by a commercial MIP solver in Step 14. This program is defined as the current RP augmented by all columns in N that have a relatively small degree of incompatibility (e.g., smaller than 7). Notice that solving this program can yield an improved solution

only if the ISUD algorithm stops with a non-integer direction in Step 12. Otherwise, solving $MIP(\mathbf{x}^r)$ is useless and can be omitted. If the computed solution $\bar{\mathbf{x}}^r$ of $MIP(\mathbf{x}^r)$ differs from the current solution $\mathbf{x}^r$, then $\bar{\mathbf{x}}^r$ becomes the current solution and ISUD is applied again in Step 17 to try to improve this new solution considering all columns in N and not only a subset of it like in $MIP(\mathbf{x}^r)$. The cost improvement realized in this iteration is again checked in Step 18. If this improvement is deemed insufficient, the number of consecutive iterations where the algorithm failed to yield a sufficiently large cost improvement is incremented by one, as well as the upper bound on the degree of incompatibility ℓ , before starting a new iteration or ending the whole solution process if $consFail$ becomes equal to $maxConsFail$, a predefined parameter value. Otherwise, the number of consecutive failures is reset to zero.

The ICG Algorithm 2 is heuristic for two reasons. First, the stopping criterion may be too restrictive. The parameters $maxDegree$ and $maxConsFail$ should be set to large unrestrictive values in an exact algorithm. Second, the disjunctive constraints (A.5) are always relaxed from the CP and solving it might not return an integer descent direction even if one exists. Consequently, in an exact algorithm, these constraints should be considered by the algorithm solving the CP.

Note that the procedure used to compute a complete dual solution α is valid to ensure the exactness of the ICG algorithm. Indeed, as shown by the next proposition, at least one variable x_j has a negative reduced cost with respect to α if the current solution is not optimal.

Proposition 1. *Let $\alpha \in \mathbb{R}^m$ be a dual solution corresponding to a solution S with cost $\sum_{j \in S} c_j$. Let S' be an improved solution ($\sum_{j \in S'} c_j < \sum_{j \in S} c_j$). Then, there exists at least one variable x_j , $j \in S' \setminus S$, that has a negative reduced cost $c_j - \sum_{i \in T} a_{ij} \alpha_i$ with respect to the dual solution α .*

Proof (by contradiction). Assume that all variables x_j , $j \in S' \setminus S$, have a non-negative reduced cost. In this case, we have

$$c_j \geq \sum_{i \in T} a_{ij} \alpha_i, \quad \forall j \in S' \setminus S \Rightarrow \sum_{j \in S' \setminus S} c_j \geq \sum_{i \in T} \sum_{j \in S' \setminus S} a_{ij} \alpha_i \quad (4.13)$$

$$c_j = \sum_{i \in T} a_{ij} \alpha_i, \quad \forall j \in S \setminus S' \Rightarrow \sum_{j \in S \setminus S'} c_j = \sum_{i \in T} \sum_{j \in S \setminus S'} a_{ij} \alpha_i. \quad (4.14)$$

Given that the sum of the columns in $S' \setminus S$ is compatible with the columns in $S \setminus S'$, it ensues that

$$\sum_{i \in T} \sum_{j \in S' \setminus S} a_{ij} \alpha_i = \sum_{i \in T} \sum_{j \in S \setminus S'} a_{ij} \alpha_i. \quad (4.15)$$

From (4.13)–(4.15), we deduce that $\sum_{j \in S' \setminus S} c_j \geq \sum_{j \in S \setminus S'} c_j$, which contradicts the fact that S' is a better solution than S . Hence, there must exist a variable x_j , $j \in S' \setminus S$, with a negative reduced cost. $\square$

Let us highlight the differences between the ICG algorithm and the standard CG algorithm. The main one is that, in the ICG algorithm, the RMP is solved at each iteration by ISUD to find an integer solution while, in the CG algorithm, the RMP is a linear program that is solved by a linear programming solver. The CG algorithm yields a lower bound and needs to be embedded in a branch-and-cut framework to derive integer solutions. In both algorithms, the subproblem generates negative reduced cost columns that have the potential to improve the current RMP solution. To do so, it requires a complete dual solution to the RMP, i.e., with a dual value for each task in T . Because decomposition is used to solve the RMP in the ICG algorithm, such a dual solution is not directly available as in the standard CG algorithm. It is rather computed by solving a linear system of equations that exploits the current solution S . Finally, the proposed ICG algorithm relies on two heuristic stopping criteria, while standard CG stops when no more negative reduced cost columns can be generated.

The main advantage of the ICG algorithm is to benefit from the strengths of the ISUD algorithm for solving the RMP, namely, i) to exploit the RMP degeneracy in order to easily find integer descent directions and ii) to compute at almost every iteration an improved solution even if the CP is not solved at optimality at each iteration. This allows to compute a sequence of improving integer solutions and stop the algorithm whenever the quality of the current solution is satisfactory (assuming that a lower bound is previously computed to assess this quality).

In the ICG algorithm, CG is applied to generate the variables of the SPP model (4.1)–(4.3). Standard CG uses the dual solution of the current RMP to generate negative reduced cost variables that can improve the current objective value, converging towards a lower bound. No effort is made to improve the current upper bound on the optimal value of the SPP. The proposed ICG algorithm is designed to do so. Indeed, it relies on a dual solution corresponding to the current integer solution which favors the generation of columns that can improve this integer solution.

Finally, note that the columns accumulated while solving the subproblem with different dual solutions lead to an optimal solution even if these dual solutions are not necessarily of good quality. It is well known that poor-quality dual solutions have a negative impact on the convergence of standard CG algorithms. Given that the ICG algorithm can handle a very large number of columns in the RMP through the column pool and the multi-phase strategy, the quality of the dual solution has less impact on the algorithm convergence when

a large number of columns is generated at each iteration. A possibility (not implemented) to further restrict the impact of the dual solution quality would be to solve in parallel several subproblems defined with different dual solutions.

4.4.2 An acceleration strategy

To speed up the solution process, we propose to generate whenever possible more than one integer descent direction each time that the CP is solved in Step 3 of Algorithm 1 when called from Steps 12 and 17 of Algorithm 2. Multiple directions can easily be handled in the RP if they are orthogonal. Therefore, after finding a first integer descent direction $\mathbf{d}_1$ defined according to (4.12), all variables λ_ℓ , $\ell \in S$, taking a positive value in the current solution of the CP is removed from the CP as well as all variables v_j , $j \in I_S$, such that $\lambda_\ell \mathbf{A}_\ell^\top \mathbf{A}_j \neq 0$ for at least one $\ell \in S$. In this way, any new solution to the CP gives a new direction that is orthogonal to $\mathbf{d}_1$. The updated CP is then solved. If it returns another integer descent direction $\mathbf{d}_2$, further variables are removed from the CP before solving it again, ensuring that only directions orthogonal to $\mathbf{d}_1$ and $\mathbf{d}_2$ can be generated. This process repeats until no integer descent direction can be found. When all directions are found, the current integer solution to the RP is updated before starting a new ISUD iteration. Generating multiple directions when solving a CP reduces the total number of ISUD iterations and, therefore, the total time to compute the degree of incompatibility of the generated columns and to build the CP at each iteration. Furthermore, given that variables are removed from the CP when each direction is found, the updated CP can be solved more rapidly. Computational results reported in the next section will show the effectiveness of this strategy that we call the *Multi-Direction in the CP (MDCP) strategy*.

4.5 Computational results

In this section, we report computational results obtained by the ICG algorithm on VCSP and CPP instances (Subsections 4.5.1 and 4.5.2, respectively). These two problems are modeled as SPPs that are subject to high degeneracy. The ICG algorithm is compared to two well-known dual-fractional heuristics (see, e.g., Joncour *et al.* [37]) that are based on CG, namely, a diving heuristic (DH) and a restricted master heuristic (RMH). DH is a branch-and-price heuristic that explores a single branch of the search tree. At each node where solving the MP results in a fractional-valued solution, it fixes to one the variable with the largest fractional value. The process stops when the solution of the MP is integer. RMH consists of solving by column generation the MP at the root node. Integrality requirements are then added on the variables contained in the last RMP. This integer RMP is then solved by a commercial MIP

solver. It should be noted that DH and RMH do not guarantee finding an integer solution. Indeed, as there is no backtrack in the search tree with DH, the algorithm may end up with an infeasible problem at the bottom of the branch explored. However, this situation did not occur in our computational experiments. For RMH, the subset of columns generated in the root node may simply not contain any feasible integer solution. This case was encountered for some of the large VCSP instances.

Based on preliminary experiments, we set the parameter values of the ICG Algorithm 2 as follows: $minImp = 0.0025$, $maxConsFail = 9$, and $maxDegree = 7$. Furthermore, the multi-phase strategy invoked in the ISUD Algorithm 1 executed the sequence of phases $k \in \{1, 2, \dots, 5\}$, stopping thus before proving optimality. A sensitivity analysis on the values of these parameters is conducted in Subsection 4.5.3.

All our tests were performed on a Linux machine equipped with four Intel Xeon E3-1226 processors clocked at 3.3GHz. All CPs, RMPs and mixed integer programs are solved by the IBM CPLEX commercial solver (version 12.4) using up to four threads. For both VCSP and CPP, there are multiple column generation subproblems that are SPPRCs. They are solved by dynamic programming using the Boost library, version 1.54.

4.5.1 VCSP results

The VCSP is one of the important planning problems faced by transit companies. It consists of assigning simultaneously buses to bus trips and drivers to tasks which are defined by dividing each bus trip into segments linking consecutive possible driver relief points. In fact, for each bus trip, there is one task for each segment it contains, one task ensuring that a bus reaches the beginning of the trip and another ensuring that a bus leaves after the end of the trip. Traditionally, for large-sized instances, the problem has been tackled in two steps, assigning the buses first and the drivers afterwards. Several exact and heuristic algorithms have been developed to solve different variants (one or several depots, homogeneous or heterogeneous fleet, various driver schedule types, etc.) of the integrated problem (see Ibarra-Rojas *et al.* [38]). We consider the single-depot, homogenous-fleet variant addressed by Haase *et al.* [33] who devised the first branch-and-price algorithm for the VCSP. We use their model, except that we omit the non-set-partitioning constraints which are necessary to count the number of buses used. In the resulting SPP model, each variable is associated with a feasible driver schedule that specifies the sequence of tasks performed (driving along a trip segment, driving toward the beginning of a trip, or driving away from the end of a trip). Given a solution composed of driver schedules, one can easily retrieve optimal bus schedules in a post-processing step. To generate the driver schedules, there is one SPPRC subproblem

for each of the two driver schedule types. The networks underlying these subproblems and their seven resources are described in details in Haase *et al.* [33].

The random instance generator of Haase *et al.* [33] was used to generate the test instances. The size of an instance (number of tasks) is defined by the number of bus itineraries (B) and the number of relief points considered along each itinerary (R). It is equal to $B(R + 1)$. For each pair (B, R) , we generated three different instances using different seed numbers. Notice that, for the same instance number, the instances for a pair (B_1, R_1) and those for a pair (B_2, R_2) with $B_1 = B_2$ differ only by the number of relief points considered. Given that the number of reliefs in the middle of a bus trip tends to be minimized in an optimal solution, the optimal solutions of these instances are often the same. Nevertheless, the model contains a different number of constraints and the solution process varies. The set of instances is divided in two classes: small (800 tasks or less) and large (between 960 and 2000 tasks). Note that real-life instances typically include between 1000 and 2000 tasks, but may involve up to 3000 tasks.

For these instances, the initial solution $\mathbf{x}^0$ was defined in Step 1 of the ICG Algorithm 2 as an artificial one, where each task associated with a bus trip is covered by an artificial column bearing a large cost. In the initial dual solution $\boldsymbol{\alpha}^0$, each dual value associated with these tasks was then set to this large value divided by the number of tasks in the bus trip.

The computational results of some tests are reported in Tables 4.1 and 4.3 for the small and the large VCSP instances, respectively. For these first tests, the ICG algorithm did not use the MDCP strategy. In these tables, the first three columns describe the instance, namely, the number of tasks it contains (Tasks), the pair (B, R) defining it, and the instance number (No). Then, for each tested algorithm, they report the total computational time in seconds, the optimality gap in percentage between the cost of the best solution found and the linear relaxation optimal value, the number of column generation iterations (Itr), the total number of columns generated (Col), and the time spent solving the subproblems (SpT). For the RMH and ICG algorithms, the total number of integer solutions found (IntS) is also indicated. This number is not reported for DH because it is always equal to 1. Finally, the number of times that a mixed integer program was solved in Step 14 of Algorithm 2 (Mip) and the time (in seconds) spent solving these programs (MipT) are also specified for the ICG algorithm. Note that the linear relaxation optimal values are only computed for reporting the optimality gaps. The time required to compute them is not included in the total time. Note also that, for ICG, the number of columns generated does not include those that were added to the pool of columns but never considered in the RP or the CP because their degree of incompatibility remained too high.

Table 4.1 Results for the small VCSP instances

<i>Instance</i>			<i>RMH</i>						<i>DH</i>					<i>ICG</i>							
<i>Tasks</i>	(B, R)	<i>No.</i>	<i>Time</i> [†]	<i>Gap</i> [†]	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>	<i>Mip</i>	<i>MipT</i>
240	(40, 5)	1	0.9	0.0	12	2171	0.6	1	1.0	0.0	25	2714	0.8	0.7	0.0	6	9984	0.1	16	0	0.0
		2	0.6	0.0	16	1396	0.4	2	0.7	0.0	29	1582	0.6	0.3	0.0	5	7583	0.1	12	0	0.0
		3	1.0	0.0	10	1389	0.4	1	0.8	0.0	30	1886	0.7	0.6	0.0	6	8806	0.1	13	0	0.0
320	(80, 3)	1	1.2	0.0	11	2699	0.9	1	3.6	0.0	70	4766	2.9	1.1	0.0	6	9935	0.2	29	0	0.0
		2	1.1	0.0	12	2348	0.7	1	1.4	0.0	32	2658	1.2	0.7	0.0	7	8561	0.2	25	0	0.0
		3	1.4	0.0	14	2673	1.1	1	3.1	0.0	58	4284	2.5	1.2	0.0	7	10101	0.3	39	0	0.0
400	(40, 9)	1	6.8	0.0	12	4827	4.6	3	11.5	0.0	83	7804	9.9	4.5	0.0	6	35250	1.0	16	0	0.0
		2	5.8	0.0	16	3798	3.0	3	8.5	0.0	68	5912	7.0	2.8	0.0	5	24032	0.5	13	0	0.0
		3	30.8	0.0	14	3229	3.1	4	7.3	0.0	62	5022	6.3	3.8	0.0	5	27710	0.8	13	0	0.0
480	(80, 5)	1	5.9	0.0	15	4201	4.7	1	13.9	0.0	87	7545	11.6	4.5	0.0	7	21431	1.0	28	0	0.0
		2	5.2	0.0	16	3748	3.9	3	11.9	0.0	77	6421	10.0	2.6	0.0	6	17657	0.6	29	0	0.0
		3	6.9	0.0	18	3806	5.0	2	11.0	0.0	66	5527	9.5	6.1	0.0	8	64839	0.7	40	1	0.9
640	(80, 7)	1	16.5	0.0	17	6204	13.3	2	38.8	0.0	94	12036	33.6	14.5	0.0	6	33824	2.6	30	0	0.0
		2	15.3	0.0	20	5314	11.6	0	39.3	0.0	128	10255	33.2	9.7	0.0	6	37871	1.8	30	0	0.0
		3	38.5	0.0	20	6234	15.1	6	27.5	0.0	61	9147	24.2	10.9	0.0	6	43899	2.2	42	0	0.0
720	(120, 5)	1	3450.5	0.7	21	8377	22.3	5	58.5	0.0	114	15918	46.5	16.8	0.0	7	44955	2.6	63	0	0.0
		2	405.9	1.2	20	5675	12.1	4	39.8	0.0	102	10461	33.3	7.4	0.0	6	28790	1.3	40	0	0.0
		3	20.9	0.0	21	7674	16.2	1	45.7	0.0	114	12671	38.8	14.4	0.0	7	36479	1.7	62	0	0.0
800	(80, 9)	1	361.7	0.4	18	7470	27.9	5	168.2	0.0	214	25273	140.1	25.0	0.0	7	56143	5.7	29	0	0.0
		2	40.2	0.0	22	7972	28.8	2	131.0	0.0	166	20918	96.1	20.4	0.0	6	47682	4.3	30	1	1.2
		3	694.7	0.0	19	8079	32.6	4	152.9	0.0	174	25632	116.4	20.9	0.0	6	65417	3.5	40	0	0.0

†: All gaps are in percentage; all times are in seconds.

From Table 4.1, we observe that ICG finds the optimal solution for all instances in faster computational times than the other two heuristics. For the largest of these small instances, ICG can be as up to 7.3 times faster than DH and RMH. This is due to the small number of iterations performed by ICG (at most 7) compared to DH and the relatively large number of integer solutions found per iteration (on average, close to 5). Given that solving the subproblems is quite time-consuming as shown by the large proportion of the total computational time they take in DH and RMH, reducing the number of iterations in ICG significantly reduces the time spent solving subproblems and, therefore, the total computational time. RMH and DH also find an optimal solution for most instances. They fail to do so for 3 and 1 instances, respectively. It, thus, seems that omitting the non-set-partitioning constraints from the VCSP model of Haase *et al.* [33] makes the problem much easier to solve. Nevertheless, we observe that, for the largest of these small instances, the computational time required by RMH may be quite large (almost up to one hour). Finally, notice that, for these instances, ICG rarely needs to solve a mixed integer program to get a larger improvement of the objective value at a given iteration. This is also the case for the large VCSP instances as reported below.

One can remark that ICG uses much more columns than RMH and DH. To verify if this observation can explain the difference in the computational times, we ran additional tests with RMH and DH, increasing the number of columns generated at each iteration. This increase was achieved by not applying the dominance rule at the sink node of the networks and selecting a large number of the resulting columns based on their reduced cost. The results

of these tests are reported in Table 4.2. They show that, when a larger number of columns is generated (on average, the number of columns generated for DH is 1.2 times larger than the number used by ICG), RMH and DH become slower, and thus ICG remains the fastest algorithm. Notice that, in this case, an optimal solution can be found for all instances by both RMH and DH. In the following, the results for RMH and DH were obtained by generating a relatively small number of columns per iteration (as for the results in Table 4.1).

Table 4.2 RMH and DH results for small VCSP instances when generating many columns

<i>Instance</i>			<i>RMH</i>					<i>DH</i>			
<i>Tasks</i>	(B, R)	<i>No</i>	<i>Time</i> [†]	<i>Gap</i> [†]	<i>Itr</i>	<i>Col</i>	<i>IntS</i>	<i>Time</i>	<i>Gap</i>	<i>Itr</i>	<i>Col</i>
240	(40, 5)	1	2.6	0.0	9	4767	1	1.7	0.0	12	6131
		2	0.6	0.0	10	5141	6	0.6	0.0	10	5141
		3	1.2	0.0	6	3600	2	1.0	0.0	8	4664
320	(80, 3)	1	2.7	0.0	15	9000	1	2.8	0.0	15	9000
		2	1.6	0.0	10	5713	1	2.8	0.0	15	7828
		3	7.8	0.0	10	6000	2	4.7	0.0	14	8039
400	(40, 9)	1	242.7	0.0	15	9000	3	21.8	0.0	45	25210
		2	48.4	0.0	17	9901	2	12.2	0.0	39	21240
		3	6.4	0.0	15	8296	1	13.3	0.0	35	18391
480	(80, 5)	1	79.2	0.0	23	13470	1	30.8	0.0	51	28390
		2	35.1	0.0	17	10200	2	17.5	0.0	38	21916
		3	46.8	0.0	16	9600	2	27.8	0.0	46	26658
640	(80, 7)	1	4033.4	0.0	34	20400	3	143.2	0.0	135	77243
		2	2077.7	0.0	31	18517	3	70.1	0.0	77	43927
		3	4026.0	0.0	25	15000	5	82.1	0.0	77	42651
720	(120, 5)	1	54.5	0.0	40	23782	1	144.8	0.0	160	93565
		2	990.8	0.0	34	20228	3	129.0	0.0	99	57308
		3	4035.4	0.0	31	18600	3	123.0	0.0	111	60795
800	(80, 9)	1	3070.8	0.0	38	22800	1	363.3	0.0	159	90212
		2	4051.2	0.0	38	22800	3	380.4	0.0	111	64339
		3	4093.6	0.0	27	16200	2	249.3	0.0	109	64084

†: All gaps are in percentage; all times are in seconds.

For all the large VCSP instances, RMH was not able to find a feasible solution within a 1-hour time limit. Therefore, we report no results for RMH in Table 4.3. The results in this table confirm the superiority of ICG over DH. Indeed, one can observe that ICG is always faster and can yield remarkable time reductions of up to 97% for the largest tested instances (besides finding an optimal solution for all tested instances). The number of iterations performed by ICG (at most 8) is negligible compared to that achieved by DH (between 123 and 990), indicating that the large number of columns generated at each iteration is fully exploited by the ISUD algorithm when solving the RMP. This shows that strategies which aim at speeding up the ISUD algorithm and, in particular, the time required to solve the CP such as the MDCP strategy can be useful to increase the efficiency of ICG. To support this assertion, we conducted another series of experiments that consisted of solving the large VCSP instances with the ICG algorithm, but this time, applying the MDCP strategy. The results of these

Table 4.3 Results for the large VCSP instances

<i>Instance</i>			<i>DH</i>					<i>ICG</i>							
<i>Tasks</i>	(B, R)	<i>No.</i>	<i>Time</i> [†]	<i>Gap</i> [†]	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>	<i>Mip</i>	<i>MipT</i>
960	(160, 5)	1	150.4	0.0	158	23978	129.6	26.6	0.0	6	48437	6.4	92	0	0.0
		2	114.5	0.0	143	16778	92.6	24.9	0.0	7	50740	7.6	67	0	0.0
		3	110.0	0.0	123	18964	95.9	22.9	0.0	6	38647	6.8	75	0	0.0
1200	(200, 5)	1	347.4	0.0	220	34397	304.0	44.7	0.0	7	65955	13.7	103	0	0.0
		2	184.7	0.0	126	20985	162.3	52.1	0.0	8	190431	10.1	101	1	4.4
		3	242.8	0.0	183	27344	212.6	42.0	0.0	6	64578	9.5	105	0	0.0
1200	(120, 9)	1	1063.9	0.0	411	73736	784.4	101.4	0.0	6	121266	16.9	65	0	0.0
		2	748.6	0.0	334	48252	548.1	49.9	0.0	7	264412	10.6	40	1	8.5
		3	1015.5	0.0	354	53992	591.5	84.0	0.0	6	124171	15.0	58	0	0.0
1600	(160, 9)	1	2330.9	3.5	419	76463	1075.1	170.6	0.0	6	140279	25.8	89	0	0.0
		2	1703.1	0.0	331	31955	785.5	150.6	0.0	7	120979	28.1	71	0	0.0
		3	2831.5	0.0	467	57719	1306.0	144.2	0.0	7	117910	25.8	75	0	0.0
2000	(200, 9)	1	10081.6	0.0	968	296499	4650.1	263.8	0.0	7	159891	61.1	107	0	0.0
		2	9345.0	0.0	873	155980	4310.3	295.3	0.0	7	539911	55.9	100	1	20.5
		3	9169.6	0.0	990	192090	5004.7	287.2	0.0	7	160804	42.9	112	1	18.3

†: All gaps are in percentage; all times are in seconds.

Table 4.4 Results of ICG with the MDCP strategy on the large VCSP instances

<i>Instance</i>			<i>ICG</i>								
<i>Tasks</i>	(B, R)	<i>No.</i>	<i>Time</i> [†]	<i>Gap</i> [†]	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>	<i>Mip</i>	<i>MipT</i>	<i>Gain</i> [†]
960	(160, 5)	1	14.5	0.0	6	51040	3.7	92	0	0.0	45.5
		2	17.9	0.0	8	146408	5.1	49	1	3.0	28.1
		3	14.1	0.0	7	42520	3.4	74	0	0.0	38.4
1200	(200, 5)	1	38.2	0.0	8	77915	8.3	105	0	0.0	14.5
		2	31.8	0.0	7	143415	8.8	106	1	3.4	39.0
		3	26.4	0.0	6	62009	6.9	111	0	0.0	37.1
1200	(120, 9)	1	53.4	0.0	6	107706	13.4	66	0	0.0	47.3
		2	28.2	0.0	6	83243	9.2	41	0	0.0	43.5
		3	60.1	0.0	6	124428	15.9	60	0	0.0	28.5
1600	(160, 9)	1	99.7	0.0	6	160164	31.1	92	0	0.0	41.6
		2	92.8	0.0	7	123600	25.5	72	0	0.0	38.4
		3	95.7	0.0	7	134650	27.6	77	0	0.0	33.6
2000	(200, 9)	1	216.7	0.0	8	540545	57.5	110	1	24.8	17.9
		2	201.4	0.0	7	500187	58.5	100	1	20.3	31.8
		3	170.5	0.0	7	197081	44.6	117	0	0.0	40.6

†: All gaps and gains are in percentage; all times are in seconds.

experiments are reported in Table 4.4. The last column in this table specifies the relative gain in computational time obtained by applying the MDCP strategy. We observe that the MDCP strategy yields substantial time reductions varying between 14.5% and 47.3%.

4.5.2 CPP results

The CPP consists of finding least-cost crew pairings such that each flight of a given schedule is actively covered by a single pairing. A pairing is a sequence of flights performed by a crew that starts and ends at the crew base. For some of these flights, the crew may be deadheading,

i.e., the crew members travel as passengers. To be feasible, a pairing must satisfy a variety of safety regulations and labor agreement rules. The CPP is usually modeled as a SPP (each variable is associated with a feasible pairing) and solved by branch-and-price (see Desaulniers *et al.* [34]), where the pairings are generated by solving subproblems that can be modeled as SPPRCs. More details on the subproblems can be found in Saddoune *et al.* [35].

For our tests, we used five real-life CPP instances (without additional constraints) obtained from the datasets proposed by Kasirzadeh *et al.* [39]. Each instance is defined for a single aircraft fleet (D94, D95, 757, 319, or 320) and spans a single week. To model the cost function and the pairing feasibility rules, nine resources are required in the SPPRC subproblems. For the ICG Algorithm 2, an artificial solution was built as an initial solution $\mathbf{x}^0$ in Step 1: each task is covered by a single-task column that bears a very large cost. In the initial dual solution $\boldsymbol{\alpha}^0$, each dual value was set to this large cost.

Table 4.5 provides the computational results obtained on the CPP instances by the three heuristics, namely, RMH, DH, and ICG with the MDCP strategy. Its first two columns identify the instance by its name and the number of tasks it contains. Then, for each heuristic, it reports the same information as in Table 4.1.

Notice that, as for the VCSP instances, the number of column generation iterations executed by ICG is very small compared to DH. This can be explained by the large number of columns generated in each iteration. These columns are efficiently handled in the RMP through the multi-phase strategy used by the ISUD algorithm. Furthermore, the multi-phase strategy applied at the subproblem level is useful to generate columns with a low incompatibility degree, yielding a low density coefficient matrix in the CP.

Table 4.5 Results for the CPP instances

<i>Instance</i>		<i>RMH</i>							<i>DH</i>					<i>ICG</i>							
<i>No.</i>	<i>Tasks</i>	<i>Time</i> [†]	<i>Gap</i> [†]	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>		<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Col.</i>	<i>SpT</i>	<i>IntS</i>	<i>Mip</i>	<i>MipT</i>
94	424	15.0	0.05	22	17110	7.4	1		35.3	0.20	76	22217	32.6	17.2	0.14	12	32044	2.9	21	4	2.8
D95	1255	3284.9	1.13	56	100825	481.8	9		5411.4	0.21	914	313989	4875.2	2164.3	0.22	20	341246	229.1	280	8	811.0
757	1290	4109.7	0.03	74	74221	519.3	4		18019.0	0.02	1203	597469	6031.3	2174.8	0.01	15	396480	135.6	212	1	320.1
319	1293	6048.0	0.69	97	81335	684.5	8		2943.9	0.29	1268	306633	2395.8	1218.2	0.18	23	283027	167.4	278	9	530.5
320	1740	3066.6	0.10	110	96188	936.0	5		4630.1	0.10	1269	367025	3701.4	3806.4	0.05	20	328827	194.0	313	6	987.2

†: All gaps are in percentage; all times are in seconds.

One remarkable characteristic of the ICG algorithm is the large number of different integer solutions found throughout the solution process (more than 10 per iteration for the largest instances). This feature is highly desirable in the industry because it allows to stop the solution process at any time after finding a first satisfactory solution. To determine the quality of a solution, one may compute a lower bound in parallel. For the tested instances, this requires less than 50% of the time required by the ICG algorithm.

Finally, we observe that the number of times that a mixed integer program was solved in Step 14 of Algorithm 2 is small and the time devoted to solving these programs varied between 15% and 44% of the total time. Despite the large proportion of time that it may consume, this step has proven to be useful to accelerate the solution process.

Figure 4.2 depicts the cost of the current solution in function of the computational time for instance 320. Each point corresponds to a solution found during the solution process. The blue circles represent those obtained by the ISUD algorithm while the red plus signs represent those produced by solving a mixed integer program. We can observe a rapid cost decrease at the beginning of the solution process. Like traditional column generation methods, the cost decrease becomes slow towards the end. Notice that the solutions obtained by solving a mixed integer program (especially the first one) can sometimes yield a large cost decrease. We would like to point out that, for all instances, the algorithm succeeded to find a first feasible integer solution within the first few iterations of the solution process.

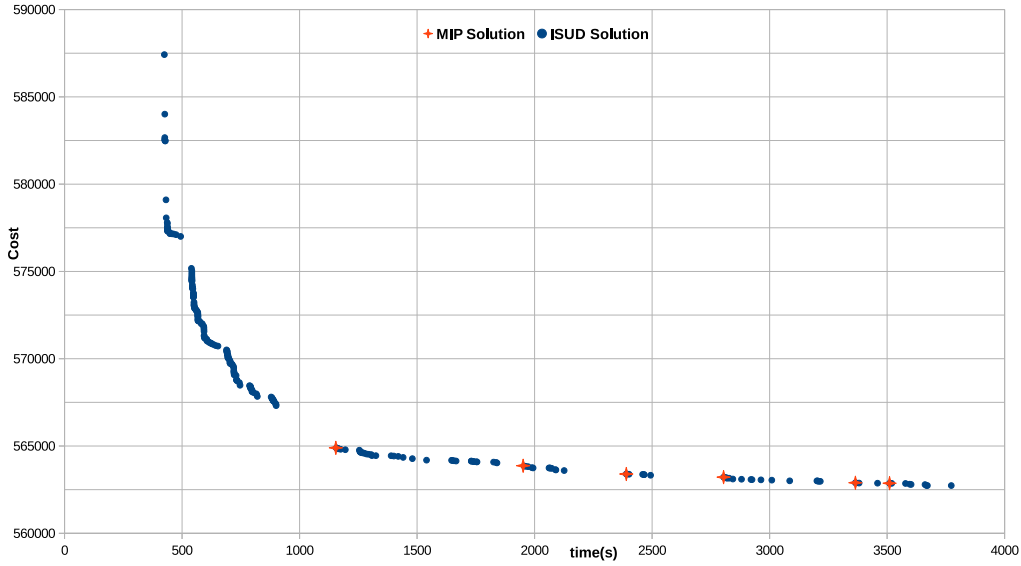


Figure 4.2 Current solution cost in function of the time (instance 320)

To conclude this section, we report in Figure 4.3 the number of pairings selected in the final solution that have already been generated at each iteration of the solution process for both DH and ICG. We observe that much more selected pairings are generated per iteration with ICG. This is not surprising given that ICG generates much more columns per iteration than DH, but also that the dual solutions used to generate columns in ICG correspond to the current integer solution and increase the chances of generating columns that can be part of better solutions.

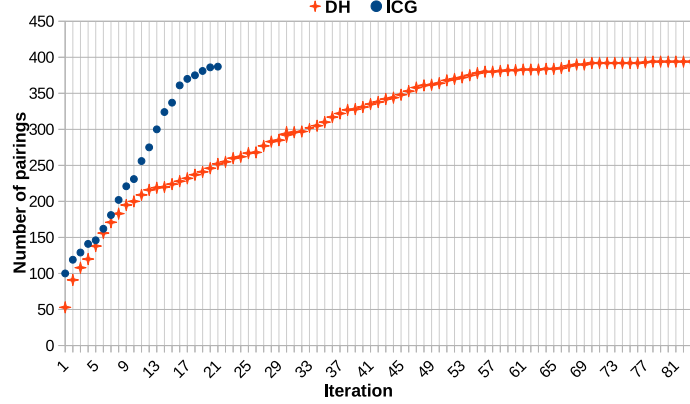


Figure 4.3 Number of pairings from the final solution already generated at each iteration (instance 320)

4.5.3 Sensitivity analysis of the ICG parameters

In this section, we provide computational results to assess the sensitivity of the ICG algorithm with respect to the values assigned to three parameters $minImp$, $maxConsFail$, and $maxDegree$. Recall that, for the results reported above, they were set as follows: $minImp = 0.0025$, $maxConsFail = 9$, and $maxDegree = 7$. We ran new experiments on the CPP instances where the value of a single parameter was changed at a time. For each parameter, we tested two new values: one smaller and one larger. These new values are $minImp \in \{0.0015, 0.003\}$, $maxConsFail \in \{7, 11\}$, and $maxDegree \in \{5, 9\}$. The results of these tests are reported in Table 4.6. For each of the six new parameter settings and each instance, we report in the top part of this table the total computational time and the variation (in percentage) with respect to the results obtained with the default setting. In the bottom part, we report the optimality gaps and the corresponding variations. The last column provides the average of the variations over all instances.

From these results, we observe that, in general, the computational times are relatively stable. Parameter $maxConsFail$ seems to have the largest impact on the computational time, which makes sense. Indeed, increasing its value increases the total number of iterations performed by the ICG algorithm unless the maximum degree of incompatibility is reached earlier, which is rarely the case. As for the gaps, we can also say that they are quite stable with relatively small variations for the instances 94, D95 and 319. Large variations are observed for instances 757 and 320 because the corresponding gaps are very small (0.01 and 0.05, respectively). Not surprisingly, the best gaps are obtained when increasing the value of $maxConsFail$, yielding a larger number of iterations and more chance to find better solutions. Setting $minImp$ to 0.003 produces a better gap for instance 319 and the same gap for the others. This may

Table 4.6 Sensitivity analysis results for the CPP instances

(minImp, maxConsFail, maxDegree)		<i>Instance</i>					<i>Average</i>
		94	D95	757	319	320	
(0.0025, 9, 5)	<i>Time</i> [†]	20	2700	1948	1260	2942	2.5
	<i>Var.</i> [†]	17.6	24.8	-10.4	3.4	-22.7	
(0.0025, 9, 9)	<i>Time</i>	22	2631	2015	1642	3719	15.2
	<i>Var.</i>	29.4	21.6	-7.4	34.8	-2.3	
(0.0025, 7, 7)	<i>Time</i>	15	2069	1736	1173	3482	-9.7
	<i>Var.</i>	-11.8	-4.4	-20.2	-3.7	-8.5	
(0.0025, 11, 7)	<i>Time</i>	25	3226	2207	1473	4042	24.9
	<i>Var.</i>	47.1	49.1	1.5	20.9	6.2	
(0.0015, 9, 7)	<i>Time</i>	21	2417	2024	1435	3995	10.2
	<i>Var.</i>	23.5	11.7	-6.9	17.8	5.0	
(0.003, 9, 7)	<i>Time</i>	26	2226	1992	1112	3324	5.2
	<i>Var.</i>	52.9	2.9	-8.4	-8.7	-12.7	
(0.0025, 9, 5)	<i>Gap</i> [†]	0.14	0.25	0.02	0.17	0.07	29.6
	<i>Var.</i>	0.0	13.6	100.0	-5.6	40.0	
(0.0025, 9, 9)	<i>Gap</i>	0.14	0.22	0.02	0.19	0.05	21.1
	<i>Var.</i>	0.0	0.0	100.0	5.6	0.0	
(0.0025, 7, 7)	<i>Gap</i>	0.17	0.23	0.03	0.19	0.08	58.3
	<i>Var.</i>	21.4	4.5	200.0	5.6	60.0	
(0.0025, 11, 7)	<i>Gap</i>	0.14	0.2	0.01	0.16	0.04	-8.0
	<i>Var.</i>	0.0	-9.1	0.0	-11.1	-20.0	
(0.0015, 9, 7)	<i>Gap</i>	0.14	0.23	0.02	0.19	0.07	30.0
	<i>Var.</i>	0.0	4.5	100.0	5.6	40.0	
(0.003, 9, 7)	<i>Gap</i>	0.14	0.22	0.01	0.16	0.05	-2.2
	<i>Var.</i>	0.0	0.0	0.0	-11.1	0.0	

†: All gaps and variations are in percentage; all times are in seconds.

be counterintuitive because increasing the value of this parameter increases the number of iterations that are considered as a failure and may thus reduce the overall number of iterations performed. On the other hand, this parameter also controls when a mixed integer program is solved at Step 14 of the ICG algorithm. When its value is larger, such a program is solved more often which may help finding a better solution.

4.6 Conclusion

In this paper, we introduced the ICG algorithm which combines column generation and the recent ISUD algorithm for solving SPPs involving a very large number of variables. This primal algorithm generates a sequence of integer solutions with decreasing costs until reaching an optimal or near-optimal solution. It benefits from generating a large number of columns at each column generation iteration and exploits at each iteration a dual solution that corresponds to the current integer solution and favors the generation of columns that can

be part of improved integer solutions. Our computational experiments on VCSP and CPP instances involving up to 2000 set partitioning constraints showed that ICG outperforms two popular column generation heuristics, producing for almost all instances better quality solutions in less computational times. For the largest VCSP instances, ICG can yield time reductions as large as 97%.

Several research avenues can be explored in the future. One of them consists of speeding up the proposed ICG algorithm by considering better columns in the CP, either by influencing the generation of the columns from the subproblems or by better selecting them from the pool of columns. Machine learning tools may be helpful to achieve this goal. Another important research direction is to generalize the ICG algorithm for solving SPPs with additional constraints.

CHAPITRE 5 ARTICLE 2: INTEGRAL COLUMN GENERATION FOR SET PARTITIONING PROBLEMS WITH SIDE CONSTRAINTS

Cet article a été soumis à la revue INFORMS Journal on Computing sous le titre "Integral Column Generation for Set Partitioning Problems with Side Constraints". Les auteurs sont Adil Tahir, Guy Desaulniers et Issmail El Hallaoui.

5.1 Introduction

The set partitioning problem (SPP) is one of the fundamental models used, in particular, for vehicle routing and crew scheduling. In a generic way, these problems can be stated as finding a least-cost set of paths in a suitable network such that they cover a set of tasks exactly once each. A task can be a customer to visit or a flight to cover, whereas a path represents a vehicle route or a bus driver/aircrew schedule. For large-scale problem instances, these paths are generated by a column generation (CG) technique because it is impossible to enumerate all of them a priori. Side constraints such as vehicle or pilot availability are often added to the SPP, yielding the SPP with side constraints (SPPSC) that we consider in this paper.

Let $\mathbf{T} = \{1, \dots, m\}$ be the set of tasks (associated with the set partitioning constraints), $\mathbf{H}$ the set of side constraint indices, and $\mathbf{N} = \{1, \dots, n\}$ the set of paths (columns). For each column $j \in \mathbf{N}$, define a binary variable x_j that is equal to 1 if column j with cost c_j is part of the solution and 0 otherwise. Furthermore, for each column $j \in \mathbf{N}$ and task $i \in \mathbf{T}$, let a_{ij} be a binary parameter equal to 1 if column j covers task i and 0 otherwise. These coefficients a_{ij} define the set partitioning constraint matrix $\mathbf{A} = (A_j)_{j \in \mathbf{N}} = (a_{ij})_{i \in \mathbf{T}, j \in \mathbf{N}}$. For each $j \in \mathbf{N}$, we assume that column A_j covers at least one task, i.e., there exists $i \in \mathbf{T}$ such that $a_{ij} = 1$. Finally, the coefficient q_{hj} , $j \in \mathbf{N}$, $h \in \mathbf{H}$, denotes the contribution of variable x_j to the side constraint h , whose right-hand side is b_h .

Given this notation, the SPPSC can be written as the following integer program:

$$(\mathbb{P}) \quad \min_x \quad \sum_{j \in \mathbf{N}} c_j x_j \quad (5.1)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{N}} a_{ij} x_j = 1, \quad \forall i \in \mathbf{T} \quad (5.2)$$

$$\sum_{j \in \mathbf{N}} q_{hj} x_j \leq b_h, \quad \forall h \in \mathbf{H} \quad (5.3)$$

$$x_j \in \{0, 1\}, \quad \forall j \in \mathbf{N}. \quad (5.4)$$

The objective function (5.1) aims at minimizing the total cost of the selected columns. The set partitioning constraints (5.2) ensure that each task is covered exactly once. The problem side constraints are expressed by (5.3). Note that we will focus only on less-than-or-equal-to side constraints but the proposed theory and methodology remain valid for equalities and greater-than-or-equal-to inequalities. Finally, binary requirements on the variables are imposed through (5.4). The SPP model is given by (5.1), (5.2), and (5.4).

In the rest of this paper, we denote by $\mathcal{F}_{SPPSC}$, $\mathcal{F}_{SPPSC\mathcal{LR}}$ and $\text{conv}(\mathcal{F}_{SPPSC})$ the feasible domains of the SPPSC model $\mathbb{P}$, its linear relaxation, and the convex hull of $\mathcal{F}_{SPPSC}$, respectively. Similarly, we write $\mathcal{F}_{SPP}$, $\mathcal{F}_{SPP\mathcal{LR}}$ and $\text{conv}(\mathcal{F}_{SPP})$ for the SPP, i.e., model $\mathbb{P}$ without the side constraints (5.3).

Because it includes the side constraints (5.3), model $\mathbb{P}$ does not possess an interesting property of the SPP model, called *quasi-integrality*. This property implies that, from any integer extreme point of $\mathcal{F}_{SPP\mathcal{LR}}$, there exists a so-called *integer* path along the edges of $\mathcal{F}_{SPP\mathcal{LR}}$ that visits only integer extreme points with decreasing costs and leads to an optimal solution. Consequently, the side constraints (5.3) add an extra difficulty to the SPP, which is already known for its high degeneracy. This difficulty mainly hinders the primal solution algorithms which aim at finding a sequence of integer solutions with decreasing costs that ends with an optimal or a quasi-optimal solution. Indeed, these algorithms can get stuck in a local minimum when the side constraints cut all integer paths between the current solution and all optimal ones.

The goal of this paper is to develop a new primal algorithm that can efficiently solve the SPPSC and must, therefore, overcome high degeneracy and the loss of the quasi-integrality property. To achieve this objective, we add artificial edges to $\mathcal{F}_{SPPSC\mathcal{LR}}$ to create new integer paths that can be traversed to reach optimal solutions. Furthermore, we generalize the integral CG algorithm (ICG) of Tahir *et al.* [40] that was designed for the SPP and combines CG and the integral simplex using decomposition algorithm (ISUD [6]), a recent primal algorithm which exploits degeneracy instead of suffering from it. Because ISUD decomposes the problem into a reduced problem (RP) and a complementary problem (CP), we study different variants of this improved ICG algorithm, denoted I²CG, where the side constraints are handled in the RP, the CP or in both subproblems. These I²CG variants are tested on real-world instances of the airline crew pairing problem (CPP) involving up to 1761 constraints and benchmarked against two popular CG heuristics. Our computational results show that, when compared to these two heuristics, I²CG can compute better-quality solutions in less than 35% of the computational time on average.

This paper is organized as follows. In Section 5.2, we review the literature on the most

commonly used dual and primal algorithms for solving the SPP or the SPPSC before summarizing our main contributions. In Section 5.3, we study, via an example, the effect of the side constraints on $\mathcal{F}_{SPP\mathcal{L}\mathcal{R}}$ and introduce a new quasi-integrality property that is exploited by the proposed algorithm. Then, in Section 5.4, we describe two approaches to handle the side constraints and present theoretical results to support the second one. Next, I²CG and its variants are detailed in Section 5.5. Finally, computational results on CPP instances are reported in Section 5.6 before drawing conclusions in Section 5.7.

5.2 Literature review and contributions

The SPP and the SPPSC can be tackled with several solution algorithms. Letchford and Lodi [12] have divided these algorithms into three classes: dual integral, dual fractional, and primal algorithms. Dual integral algorithms maintain integrality and dual feasibility throughout the solution process and strive to reach primal feasibility. Because this class contains a single algorithm, that of Gomory [41], we focus on the other two classes below.

5.2.1 Dual fractional algorithms

Dual fractional algorithms relax the binary requirements but maintain at each iteration primal and dual feasibility. Cutting plane algorithms as well as branch-and-bound and branch-and-cut algorithms fall in this class. For solving SPPs or SPPSCs involving a very large number of variables, branch-and-price (BP, Barnhart *et al.* [10], Desaulniers *et al.* [42]) is one of the best-known dual fractional algorithms. BP is a branch-and-bound algorithm where the linear relaxation at each node of the search tree is solved by CG. CG is an iterative method that solves a linear program by decomposing it into a restricted master problem (RMP) and a pricing subproblem (PS). For the SPPSC, the RMP is the linear relaxation of (5.1)-(5.3) restricted to a subset of its variables, whereas the PS can be, for example, a shortest path problem with resource constraints. At each iteration of CG, the RMP is first solved by a linear programming solver to provide a primal and a dual solution $\pi = ((\pi_t)_{t \in \mathbf{T}}, (\pi_h)_{h \in \mathbf{H}}) \in \mathbb{R}^{|\mathbf{T}|} \times \mathbb{R}_-^{|\mathbf{H}|}$. Then, the PS is solved by a specialized algorithm, e.g., a labeling algorithm, to find variables x_j that are not in the RMP and that have a negative reduced cost $\bar{c}_j = c_j - \sum_{i \in \mathbf{T}} \pi_i a_{ij} - \sum_{h \in \mathbf{H}} \pi_h q_{hj}$. These variables (columns) are added to the RMP which is re-optimized to start a new iteration. When no variables are generated, CG stops and the current RMP solution is optimal for the relaxed SPPSC.

To improve the effectiveness of branch-and-price algorithms, several research works have been conducted to reduce the impact of factors influencing the convergence of the standard

CG (see [16]). With the same purpose, El Hallaoui *et al.* [17, 18] introduced the dynamic constraint aggregation algorithm (DCA) and the multi-phase DCA (MPDCA). These two algorithms reduce the impact of the degeneracy in the RMP and the number of fractional variables in the linear relaxation solutions by lowering the number of set partitioning constraints in the RMP. Bouarab *et al.* [20] generalized this work by combining CG and the improved primal simplex algorithm (IPS, [19]), which is effective against degeneracy for general linear programs.

5.2.2 Primal algorithms

Unlike dual algorithms, primal algorithms maintain the feasibility of all constraints (5.2)-(5.3), including the binary requirements. They stop when optimality is reached. In other words, these algorithms search for a decreasing sequence of integer solutions leading to an optimal solution. For the SPP, the existence of this sequence has been proven by Balas and Padberg [1, 22]. This result is based on the *quasi-integrality* property proved by Trubin [21] which indicates that the edges of $\text{conv}(\mathcal{F}_{\text{SPP}})$ are also edges of $\mathcal{F}_{\text{SPP}\mathcal{L}\mathcal{R}}$. This property will be discussed in detail in Section 5.3. Several authors have studied the SPP to develop new algorithms that search for such a sequence of integer solutions. The algorithms proposed by Rönnberg and Larsson [4, 5] and Thompson [2] explore, by enumeration, the adjacent degenerate bases associated with extreme points. Their objective is to find a basis inducing a non-degenerate pivot that improves the solution. These highly combinatorial algorithms are not effective for large-scale SPP instances, mainly due to severe degeneracy.

In the same perspective, Zaghrouti *et al.* [6] proposed a new primal algorithm, called ISUD. The latter decomposes the SPP into a RP and a CP. The RP is defined as the SPP restricted to a subset of its variables and constraints (including integrality). It searches for an improved solution in the vector subspace generated by the columns of the current integer solution support. In the complementary subspace, the CP looks for descent directions qualified as *integer*.

Definition 3. A descent direction $d \in \mathbb{R}^{|\mathbf{N}|}$ is said to be minimal with respect to a solution x^0 if and only if (i) pivoting on all the variables x_j , $j \in \{j \in \mathbf{N} | d_j > 0\}$, allows to reach an extreme point adjacent to x^0 ; (ii) pivots on any strict subset of these variables are degenerate. In addition, d is said to be integer if it leads to an integer solution (i.e., $x^0 + d$ is integer); otherwise, it is said to be fractional.

Like other primal algorithms, ISUD starts with an integer solution x^0 . Denote by $\mathcal{S} = \text{supp}(x^0) = \{j \in \mathbf{N} | x_j^0 \neq 0\}$, the column index set of the solution support. By breach of

terminology, we say that $\mathcal{S}$ is the solution x^0 and that columns A_j , $j \in \mathcal{S}$, are in solution x^0 or in $\mathcal{S}$. The other columns A_j , $j \in \mathbf{N} \setminus \mathcal{S}$, are either compatible or incompatible according to the following compatibility definition (Elhallaoui *et al.* [19]).

Definition 4. A subset $\mathcal{U}$ of $\mathbf{N}$ is said to be compatible with $\mathcal{S}$, or simply compatible, if there exist two vectors $v \in \mathbb{R}_+^{|\mathcal{U}|}$ and $\lambda \in \mathbb{R}^{|\mathcal{S}|}$ such that $\sum_{j \in \mathcal{U}} v_j A_j = \sum_{l \in \mathcal{S}} \lambda_l A_l$. The combination of columns, possibly a singleton, $\sum_{j \in \mathcal{U}} v_j A_j$ is also qualified as compatible.

Let $\mathbf{C}_\mathcal{S}$ be the index set of the individually compatible columns (including those in $\mathcal{S}$) and $\mathbf{I}_\mathcal{S}$ the index set of the incompatible columns. Note that $\mathbf{I}_\mathcal{S} \cap \mathbf{C}_\mathcal{S} = \emptyset$. The compatible columns are considered in the RP, whose mathematical formulation is as follows:

$$(RP1) \quad \min_x \quad \sum_{j \in \mathbf{C}_\mathcal{S}} c_j x_j \quad (5.5)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{C}_\mathcal{S}} a_{ij} x_j = 1, \quad \forall i \in \mathbf{T}' \quad (5.6)$$

$$x_j \in \{0, 1\}, \quad \forall j \in \mathbf{C}_\mathcal{S} \quad (5.7)$$

where $\mathbf{T}' \subseteq \mathbf{T}$ is the index set of a maximal subset of linearly independent rows of matrix $\mathbf{A}$ when restricted to the columns in $\mathbf{C}_\mathcal{S}$ (e.g., subset of the first rows covered by each column A_j , $j \in \mathcal{S}$). RP1 can easily be solved using a commercial mixed-integer programming solver (e.g., Cplex). Note that the columns A_j , $j \in \mathcal{S}$, forms a non-degenerate basis of the constraint matrix of RP1 and, if there exists a variable x_j , $j \in \mathbf{C}_\mathcal{S} \setminus \mathcal{S}$, with a negative reduced cost with respect to the corresponding dual solution, then pivoting this variable into the basis yields an improved solution. In this case, the sets $\mathcal{S}$ and $\mathbf{C}_\mathcal{S}$ are updated and this process is repeated as long as RP1 finds an improved solution. Otherwise, a CP is solved to find a minimal descent direction $d \in \mathbb{R}^{|\mathbf{N}|}$. Algebraically, d is integer if the combination of incompatible columns A_j , $j \in \mathcal{U}$, is compatible with $\mathcal{S}$ and composed of disjoint columns according to the following definition.

Definition 5. Let $\mathcal{U} \subset \mathbf{N}$ be a subset of column indices. The columns A_j , $j \in \mathcal{U}$, are said to be disjoint if and only if $A_{j_1}^\top A_{j_2} = 0$, $\forall j_1, j_2 \in \mathcal{U}$, $j_1 \neq j_2$.

Let v_j , $j \in \mathbf{I}_\mathcal{S}$, be the variables defining the weights of the incompatible columns in a linear combination $\sum_{j \in \mathbf{I}_\mathcal{S}} v_j A_j$. Let λ_l , $l \in \mathcal{S}$, be the variables defining the weights of the columns in a linear combination $\sum_{l \in \mathcal{S}} \lambda_l A_l$. Let $\mathbf{F}_\mathcal{S} = \{(j_1, j_2) \in \mathbf{I}_\mathcal{S} \times \mathbf{I}_\mathcal{S} \mid A_{j_1}^\top A_{j_2} \neq 0, j_1 \neq j_2\}$ be the set of all pairs of non-disjoint incompatible column indices. The CP, denoted CP1 to indicate its first version, is formulated as follows:

$$(CP1) \quad z^{CP1} = \min_{v, \lambda} \quad \sum_{j \in \mathbf{I}_S} c_j v_j - \sum_{l \in \mathcal{S}} c_l \lambda_l \quad (5.8)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{I}_S} a_{ij} v_j - \sum_{l \in \mathcal{S}} a_{il} \lambda_l = 0, \quad \forall i \in T \quad (5.9)$$

$$\sum_{j \in \mathbf{I}_S} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l \lambda_l = 1 \quad (5.10)$$

$$v_j \geq 0, \quad \forall j \in \mathbf{I}_S \quad (5.11)$$

$$v_{j_1} v_{j_2} = 0, \quad \forall (j_1, j_2) \in \mathbf{F}_S. \quad (5.12)$$

The objective function (5.8) aims at minimizing the reduced cost of the chosen linear combination of incompatible columns $\sum_{j \in \mathbf{I}_S} v_j A_j$. Constraints (5.9) ensure that this linear combination is compatible with $\mathcal{S}$. Involving the non-negative scalars α_j , $j \in \mathbf{I}_S$, and β_l , $l \in \mathcal{S}$, the normalization constraint (5.10) bounds the problem (otherwise, CP1 would be unbounded when an integer descent direction exists). Constraints (5.12) ensure that the selected incompatible columns A_j are pairwise disjoint. If $z^{CP1} < 0$, then an integer descent direction is found and the solution $\mathcal{S}$ is improved by replacing the columns A_l , $l \in \mathcal{S}$, such that $\lambda_l > 0$ with the columns A_j , $j \in \mathbf{I}_S$, such that $v_j > 0$ (i.e., $d = (d_j)_{j \in \mathbf{N}}$ with $d_j = 1$ if $j \in \mathbf{I}_S$ and $v_j > 0$, $d_j = -1$ if $j \in \mathcal{S}$ and $\lambda_j > 0$, and $d_j = 0$ otherwise). It should be noted that in practice, the constraints (5.12) are relaxed from CP1 to yield a linear program. As shown by Zaghroui *et al.* [6], this relaxed CP1 often finds integer descent directions. When this is not the case, branching can be done to cut the fractional direction. In the following, we keep using CP1 to denote its relaxed version.

Alternating between solving RP1 and CP1 allows ISUD to find a decreasing sequence of integer solutions leading to an optimal solution. The computational experiments carried out by Zaghroui *et al.* [6] on 90 instances of the vehicle and crew scheduling problem (VCSP) and the CPP involving up to 1600 constraints and 500,000 variables (generated a priori) have shown the effectiveness of ISUD.

However, ISUD has two major limitations. First, CP1 may find fractional descent directions. Second, ISUD can only reach at each iteration the extreme points that are adjacent to the current integer solution. To overcome the first limitation, Rosat *et al.* [23, 24] proposed to add cuts to CP1 to exclude fractional directions. Finding these cuts is usually time-consuming and several of them may be required to derive an integer direction. These authors also proved that the integrality of the directions found by CP1 is influenced by the weights of the variables in the normalization constraint (5.10).

Alternatively, Zaghrouti *et al.* [7] developed a variant of ISUD, called ZOOM, that zooms on the neighborhood of a fractional direction to look for an improved integer solution. To do so, it adds a subset of incompatible columns to the RP which is then solved by a commercial mixed-integer programming solver. If no better integer solution is found, the neighborhood is enlarged and the RP is solved again. The process is repeated until finding a better solution. In the remainder of the paper, we use the term *zooming* to refer to the search for an integer direction in the neighborhood of a fractional one.

To overcome the second limitation of ISUD, Zaghrouti *et al.* [8] proposed a second variant of ISUD, named I²SUD, where an artificial variable is added to the SPP. Let $\mathbb{K}$ be an SPP, x^0 the current integer solution of $\mathbb{K}$ and $\mathcal{S} = \{j \in \mathbf{N} \mid x_j^0 \neq 0\}$. Let $\mathbb{K}'$ be the new SPP created from $\mathbb{K}$ by adding the artificial variable x_{n+1} , such that $x' = \begin{bmatrix} x & x_{n+1} \end{bmatrix}$, $c' = \begin{bmatrix} c & c_{\mathcal{S}} \end{bmatrix}$, $\mathbf{A}' = \begin{bmatrix} \mathbf{A} & e \end{bmatrix}$ and $\mathbf{N}' = \mathbf{N} \cup \{n+1\}$. The cost $c_{\mathcal{S}}$ of variable x_{n+1} is equal to the cost of the current solution ($c_{\mathcal{S}} = \sum_{l \in \mathcal{S}} c_l$). The e column, whose components are equal to 1, is the result of aggregating the columns in $\mathcal{S}$. So, the current solution x^0 can also be represented by the following artificial solution: $x_{n+1} = 1$ and $x_j = 0, \forall j \in \mathbf{N}$. In this case, the relaxed CP model of $\mathbb{K}'$ is as follows:

$$(CP2) \quad z^{CP2} = \min_{v, \lambda} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} c_j v_j + \sum_{l \in \mathcal{S}} c_l v_l - c_{\mathcal{S}} \lambda \quad (5.13)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} a_{ij} v_j + \sum_{l \in \mathcal{S}} a_{il} v_l - \lambda = 0, \quad \forall i \in \mathbf{T} \quad (5.14)$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l v_l = 1 \quad (5.15)$$

$$v \geq 0, \lambda \geq 0. \quad (5.16)$$

If $z^{CP2} < 0$, then a (possibly fractional) descent direction is found. As for CP1, the positive-valued variables $v_j, j \in \mathbf{N} \setminus \mathcal{S}$, identify the columns entering in the improved solution. The zero-valued variables $v_l, l \in \mathcal{S}$, correspond to the leaving variables. Because the directions found by CP2 are minimal with respect to the artificial solution, all extreme points in the convex hull of the feasible domain of $\mathbb{K}'$ are adjacent to this solution (see Proposition 3.2 in [8]). Consequently, CP2 can find an integer descent direction leading directly to an optimal solution of $\mathbb{K}$. The computational results reported in [8] show that I²SUD outperforms ISUD and often finds an optimal solution by solving a single CP2.

The three algorithms ISUD, ZOOM and I²SUD described above were tested on SPP instances where the columns are all generated a priori. This is generally not possible for very large-scale problem instances. For this reason, Tahir *et al.* [40] proposed a new primal algorithm, called

integral column generation (ICG), for the SPP. It is based on a three-level decomposition: RP, CP and the CG PS. At each ICG iteration, the RMP is solved using ISUD to find a sequence of integer solutions. Then, a dual solution, said to correspond to the current integer solution, is used in the PS to generate new variables with a negative reduced cost. The authors showed that, if optimality is not yet reached, then at least one entering variable identified by an integer direction has a negative reduced cost with respect to any such dual solution. Computational results on CPP and VCSP instances show that ICG outperforms two CG heuristics commonly used in practice.

5.2.3 Main contributions

This paper is a continuation of the research work on the development of primal algorithms to efficiently solve the SPP as we introduce a new algorithm to efficiently solve the SPPSC. Below, we summarize its most important contributions.

- We introduce a new property of the SPP, called *pseudo-quasi-integrality*. Then, we show with an example that the side constraints in the SPPSC invalidate the quasi-integrality and pseudo-quasi-integrality properties of the SPP. Finally, we prove that it is possible to restore the pseudo-quasi-integrality property by increasing the size of the problem.
- We formulate new RP and CP which consider the side constraints.
- We present theoretical results that characterize the integer descent directions found by the new CP.
- Based on these theoretical results, we introduce the new primal algorithm I²CG for solving the SPPSC.
- We test and compare two I²CG versions on CPP instances involving up to 1740 flights.

5.3 Side constraints and quasi-integrality properties

In this section, we first show through an example that the SPPSC is not quasi-integral. Then, we introduce a new property (pseudo-quasi-integrality) of the SPP and discuss if it applies to the SPPSC.

Let us begin by defining the quasi-integrality property.

Definition 6. *A model $\mathbb{Q}$ is said to be quasi-integral if any edge of the convex hull of its integer solutions is an edge of the linear relaxation polyhedron of $\mathbb{Q}$.*

As mentioned earlier, quasi-integrality implies the existence of an integer path between any

pair of integer extreme points of this polyhedron. Unfortunately, the side constraints in the SPPSC can invalidate this property as shown by the following example.

$$\begin{aligned}
 (\mathbb{P}_1) \quad & \min_x 12x_1 + 12x_2 + 12x_3 + 10x_4 + 10x_5 + 10x_6 + 2x_7 + 2x_8 + 2x_9 \\
 & x_1 + x_4 + x_7 + x_8 = 1 \\
 & x_1 + x_5 + x_7 + x_9 = 1 \\
 & x_2 + x_5 + x_7 + x_8 = 1 \\
 & x_2 + x_5 + x_8 + x_9 = 1 \\
 & x_3 + x_6 = 1 \\
 & x_3 + x_6 = 1 \\
 & x_3 + x_4 + x_5 \leq \frac{5}{2} \\
 & x_1 + x_2 + x_6 \leq 2 \\
 & x \in \{0, 1\}^9
 \end{aligned}$$

For this example, we present some solutions of interest in Table 5.1, where x^0 is an initial integer solution and x^4 is the unique optimal solution. Observe that these two solutions are the only feasible ones and that we do not list all the extreme points of $\mathcal{F}_{SPPSC\mathcal{LR}}$. In Table 5.1, there are: the two feasible integer solutions x^0 and x^4 , one fractional solution x^1 that is an extreme point of $\mathcal{F}_{SPPSC\mathcal{LR}}$, and two infeasible integer solutions x^2 and x^3 that would be feasible for the corresponding SPP, as shown in Figure 5.1. In this figure, the green and red nodes indicate integer and fractional extreme points of $\mathcal{F}_{SPPSC\mathcal{LR}}$, respectively. The grey nodes correspond to solutions x^2 and x^3 that are cut off by the side constraints in SPPSC. Note that two extreme points of $\mathcal{F}_{SPPSC\mathcal{LR}}$, corresponding to solutions x^i and x^j , are adjacent if and only if there exists a minimal direction d such that $x^i = x^j + d$ (see [19]).

Table 5.1 Some feasible and infeasible solutions of model $\mathbb{P}_1$

<i>Solution</i>	<i>Value</i>
$x^0 = (1, 1, 1, 0, 0, 0, 0, 0, 0)$	36
$x^1 = (0, 0, 1, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$	15
$x^2 = (1, 1, 0, 0, 0, 1, 0, 0, 0)$	34
$x^3 = (0, 0, 1, 1, 1, 0, 0, 0, 0)$	32
$x^4 = (0, 0, 0, 1, 1, 1, 0, 0, 0)$	30

Figures 5.1.i and 5.1.ii show that the edge (x^0, x^4) of $\text{conv}(\mathcal{F}_{SPPSC})$ is not an edge of $\mathcal{F}_{SPPSC\mathcal{LR}}$. We can, thus, conclude that the SPPSC is not quasi-integral because of the side constraints. In addition, there is no integer path between x^0 and x^4 in $\mathcal{F}_{SPPSC\mathcal{LR}}$. This means that, from x^0 , all the minimal descent directions that can be found by CP1 are fractional (or infeasible because it does not consider the side constraints). Therefore, ISUD can

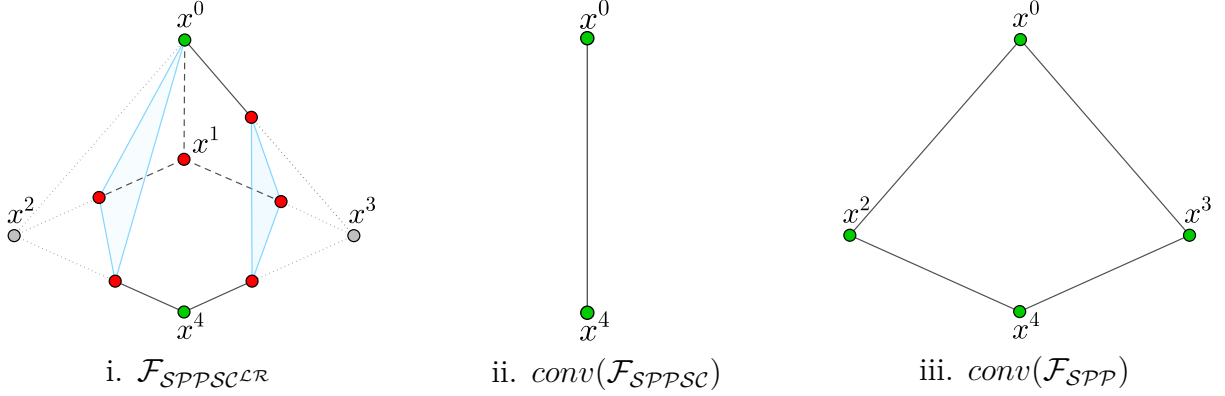


Figure 5.1 $\mathcal{F}_{SPPSC}^{LR}$, $\text{conv}(\mathcal{F}_{SPPSC})$ and $\text{conv}(\mathcal{F}_{SPP})$ of model $\mathbb{P}_1$.

get stuck in the local minimum x^0 and never reach the optimal solution x^4 .

This limitation can be overcome with the ZOOM algorithm, which can reach integer extreme points that are not necessarily adjacent to the current integer solution. However, before finding such a solution, ZOOM may need to increase the neighborhood several times, solving a relatively small binary linear program each time.

Quasi-integrality is a strong property that is not necessary in practice as primal algorithms only need to ensure the existence of an integer path between any pair of integer extreme points or, more precisely, between any initial integer solution and any optimal one. As we will show later, these paths can exist even if the problem is not quasi-integral. Let us present this requirement as the following property of a model.

Definition 7. *A model is said to be pseudo-quasi-integral when there is at least one integer path between any pair of integer extreme points of its linear relaxation polyhedron.*

As shown by the above example, the SPPSC does not possess the pseudo-quasi-integrality property. However, this property can be obtained by adding artificial edges to $\mathcal{F}_{SPPSC}^{LR}$. Indeed, inspired by I²SUD, an artificial variable x_{n+1} can be added to model $\mathbb{P}$, yielding a new model denoted $\mathbb{P}'$. The cost assigned to variable x_{n+1} corresponds to the value of the current solution $\mathcal{S}$ (i.e., $c_{\mathcal{S}} = \sum_{l \in \mathcal{S}} c_l$). The constraint coefficients of variable x_{n+1} are equal to the right-hand sides of the constraints (5.2)-(5.3), i.e., $a_{i,n+1} = 1, \forall i \in \mathbf{T}$, and $q_{h,n+1} = b_h, \forall h \in \mathbf{H}$. By construction, $x'^0 = (0, 0, \dots, 0, 1)$ is an integer solution of $\mathbb{P}'$. As shown in Section 5.4, this so-called *aggregated solution* is adjacent to all other integer extreme points of the linear relaxation polyhedron of model $\mathbb{P}'$. Therefore, it ensues the following proposition.

Proposition 2. *Model $\mathbb{P}'$ is pseudo-quasi-integral.*

Proof. Let x'^1 and x'^2 be any arbitrary pair of integer extreme points of the linear relaxation polyhedron of model $\mathbb{P}'$. Given that x'^0 is adjacent to both x'^1 and x'^2 , x'^2 can be reached from x'^1 through the integer path $x'^1 - x'^0 - x'^2$. Therefore, model $\mathbb{P}'$ is pseudo-quasi-integral. $\square$

Proposition 2 shows that, even if the side constraints in the SPPSC break the quasi-integrality and pseudo-quasi-integrality properties of the SPP, one can restore the pseudo-quasi-integrality by adding the artificial variable x_{n+1} . This variable also creates a shortcut, in the polyhedron, between the current solution and the optimal one.

5.4 Handling side constraints in ISUD

To solve the SPPSC, we develop an ICG algorithm, to be described in Section 5.5, where the RMP is solved by a new version of ISUD that can consider the side constraints. Here, we present how ISUD can be modified to handle the side constraints either in the RP or in the CP, or in both problems.

5.4.1 Handling side constraints in the RP

As mentioned earlier, the RP is formulated as a binary linear program. Therefore, the side constraints can easily be incorporated into its formulation to yield the following RP formulation:

$$(RP2) \quad \min_x \quad \sum_{j \in \mathbf{C}_S} c_j x_j \quad (5.17)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{C}_S} a_{ij} x_j = 1, \quad \forall i \in \mathbf{T}' \quad (5.18)$$

$$\sum_{j \in \mathbf{C}_S} q_{hj} x_j \leq b_h, \quad \forall h \in \mathbf{H} \quad (5.19)$$

$$x_j \in \{0, 1\}, \quad \forall j \in \mathbf{C}_S. \quad (5.20)$$

Unlike RP1, the current integer solution S does not completely fill the basis when there are side constraints in RP2. Therefore, pivoting into the basis a first negative reduced cost variable does not ensure finding a better integer solution because of possible degeneracy. Nevertheless, compared to the original model $\mathbb{P}$, RP2 considers only a subset of its variables (the compatible ones) and a subset of its partitioning constraints (5.2), resulting in bases of reduced dimension. Consequently, the linear relaxations are easier to solve and the linear relaxation solutions contain less fractional-valued variables, resulting in a potentially much smaller branch-and-bound search tree to explore.

5.4.2 Handling side constraints in the CP

As discussed in Section 5.3, CP1 cannot handle the side constraints and, therefore, needs to be redefined. To do so, we consider model $\mathbb{P}'$ and the current integer solution $\mathcal{S}$ expressed in its aggregated form $x_{n+1} = 1$, $x_j = 0$, $\forall j \in \mathbf{N}$, which we also denote $\mathcal{S}' = \{n+1\}$. The corresponding CP, denoted CP3, is formulated as follows:

$$(CP3) \quad z^{CP3} = \min_{v, \lambda} \sum_{j \in \mathbf{N} \setminus \mathcal{S}} c_j v_j + \sum_{l \in \mathcal{S}} c_l v_l - c_{\mathcal{S}} \lambda \quad (5.21)$$

$$\text{s.t.:} \quad \sum_{j \in \mathbf{N} \setminus \mathcal{S}} a_{ij} v_j + \sum_{l \in \mathcal{S}} a_{il} v_l - \lambda = 0, \quad \forall i \in \mathbf{T} \quad (5.22)$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{hj} v_j + \sum_{l \in \mathcal{S}} q_{hl} v_l - \lambda b_h \leq 0, \quad \forall h \in \mathbf{H} \quad (5.23)$$

$$\sum_{j \in \mathbf{N} \setminus \mathcal{S}} \alpha_j v_j + \sum_{l \in \mathcal{S}} \beta_l v_l = 1 \quad (5.24)$$

$$v \geq 0, \lambda \geq 0. \quad (5.25)$$

where variable λ is associated with variable x_{n+1} and always takes a positive value in any feasible solution. A solution (v, λ) to CP3 defines a descent direction $(v, -\lambda)$ if $z^{CP3} < 0$. In this case, we can deduce from this direction the following descent direction $d = (d_j)_{j \in \mathbf{N}}$ for the original problem $\mathbb{P}$:

$$d_j = \begin{cases} \frac{v_j - \lambda}{\lambda} & \text{if } j \in \mathcal{S} \\ \frac{v_j}{\lambda} & \text{if } j \in \mathbf{N} \setminus \mathcal{S} \end{cases} \quad \forall j \in \mathbf{N}. \quad (5.26)$$

One can easily verify that $c^\top d < 0$ and that $x^0 + d$ satisfies constraints (5.2) and (5.3) of $\mathbb{P}$. This direction d can, however, be integer or fractional according to Definition 3.

The following proposition characterizes the integrality of direction d in two different ways.

Proposition 3. *The following three statements are equivalent:*

1. *Descent direction d is integer;*
2. *$v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$;*
3. *For all pairs of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$, their columns A_{j_1} and A_{j_2} are disjoint.*

Proof. First, let us show that Statement 1 is true if and only if Statement 2 is true. ($\Rightarrow$) If d is integer, then, for $j \in \mathcal{S}$, $d_j = \frac{v_j - \lambda}{\lambda}$ must be in $\{0, -1\}$ because $x_j = 1$ and, for $j \in \mathbf{N} \setminus \mathcal{S}$, $d_j = \frac{v_j}{\lambda}$ must be in $\{0, 1\}$ because $x_j = 0$. In both cases, we deduce that $v_j \in \{0, \lambda\}$. ($\Leftarrow$)

If $v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$, then it is easy to verify that $d_j \in \{0, -1\}$ if $j \in \mathcal{S}$ and $d_j \in \{0, 1\}$ if $j \in \mathbf{N} \setminus \mathcal{S}$. Therefore, $x^0 + d$ is integer and, thus, d is integer.

Second, let us show that Statement 2 is true if and only if Statement 3 is true. ($\Rightarrow$) If $v_j \in \{0, \lambda\}$, $\forall j \in \mathbf{N}$, then $A_{j_1}^\top A_{j_2} = 0$ for every pair of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$. Otherwise, for one of these pairs $j_1, j_2 \in \mathbf{N}$, there would exist a task $i \in \mathbf{T}$ such that $a_{ij_1} = a_{ij_2} = 1$ and the constraint (5.22) associated with task i would be violated ($\sum_{j \in \mathbf{N}} a_{ij} v_j \geq a_{ij_1} v_{j_1} + a_{ij_2} v_{j_2} = 2\lambda \neq \lambda$). ($\Leftarrow$) Consider an index $j^* \in \mathbf{N}$ such that $v_{j^*} > 0$ and a task $i \in \mathbf{T}$ such that $a_{ij^*} = 1$. If $A_{j_1}^\top A_{j_2} = 0$ for every pair of indices $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$, then there is no other index $j' \neq j^*$ such that $v_{j'} > 0$ and $a_{ij'} = 1$. Consequently, from the constraint (5.22) associated with task i , we deduce that $\sum_{j \in \mathbf{N}} a_{ij} v_j = v_{j^*} = \lambda$. $\square$

If direction d is integer, then the corresponding descent direction $(v, -\lambda)$ suggests to replace the current aggregated solution $\mathcal{S}'$ of problem $\mathbb{P}'$ by the solution $\mathcal{S}'' = \{j \in \mathbf{N} \mid v_j > 0\}$, i.e., the columns in $\mathcal{S}''$ must be pivoted in the solution while column of index $n + 1$ must be pivoted out of it. The next proposition indicates that this direction is minimal.

Proposition 4. *Any descent direction $(v, -\lambda)$ found by CP3 which induces an integer direction d is minimal with respect to the aggregated solution $\mathcal{S}'$ of problem $\mathbb{P}'$.*

Proof. If d is integer, then $A_{j_1}^\top A_{j_2} = 0$ for every index pair $j_1, j_2 \in \mathbf{N}$ such that $j_1 \neq j_2$ and $v_{j_1} v_{j_2} > 0$. Consequently, the column of index $n + 1$ which has a coefficient of -1 for all set partitioning constraints cannot be pivoted out of the solution unless all columns in $\{j \in \mathbf{N} \mid v_j > 0\}$ are pivoted in the solution. $\square$

The following theorem states that all improved solutions to $\mathbb{P}'$ are adjacent to the current aggregated solution x'^0 in the convex hull of the feasible domain of $\mathbb{P}'$, which is denoted $\text{conv}(\mathcal{F}_{\mathbb{P}'})$.

Theorem 1. *Let x'^* be a solution to $\mathbb{P}'$ whose cost is less than the cost of x'^0 . Then, x'^* and x'^0 are adjacent extreme points in $\text{conv}(\mathcal{F}_{\mathbb{P}'})$.*

Proof. Let x^* and x^0 be the solutions corresponding to x'^* and x'^0 in problem $\mathbb{P}$. Denote by $d = x^* - x^0$ the integer descent direction that allows to move from x^0 to x^* . This direction d

can be obtained from the following solution (v, λ) to CP3 (built using Proposition 3):

$$v_j = \begin{cases} \lambda & \text{if } d_j > 0 \text{ and } j \in \mathbf{N} \setminus \mathcal{S}, \\ \lambda & \text{if } d_j = 0 \text{ and } j \in \mathcal{S}, \\ 0 & \text{otherwise} \end{cases} \quad (5.27)$$

$$\lambda = \left(\sum_{j \in \mathbf{N} \setminus \mathcal{S} \mid d_j > 0} \alpha_j + \sum_{j \in \mathcal{S} \mid d_j = 0} \beta_j \right)^{-1}. \quad (5.28)$$

According to Proposition 4, this solution yields a descent direction $(v, -\lambda)$ that is minimal. Therefore, x'^* is an extreme point of $\text{conv}(\mathcal{F}_{\mathbb{P}'})$ that is adjacent to x^0 . $\square$

Note that this theorem can easily be extended to any solution x'^* of $\mathbb{P}'$ which does not have a cost less than that of x^0 .

In contrast to CP2, CP3 considers the side constraints of the original problem $\mathbb{P}$. CP2 finds directions to explore $\mathcal{F}_{\text{SPPCR}}$ and such directions might lead to extreme points of $\mathcal{F}_{\text{SPPCR}}$ that are cut off by a side constraint of $\mathbb{P}$ (see Figure 5.1.i). The next proposition shows that this is not the case with CP3 when it returns an integer direction.

Proposition 5. *Let x be an integer extreme point of $\mathcal{F}_{\text{SPPCR}}$ that is cut off by a side constraint (5.3), say, the one indexed by $\hat{h} \in \mathbf{H}$. Then, no feasible solution (v, λ) to CP3 can yield the integer direction $d = x - x^0$ according to definition (5.26).*

Proof. Assume that there exists such a solution (v, λ) . Because d is integer, we can deduce from (5.26) and Proposition 3 that $x_j = \frac{v_j}{\lambda}$, $\forall j \in \mathbf{N}$. If x violates side constraint (5.3) for index $\hat{h}$, then

$$\begin{aligned} & \sum_{j \in \mathbf{N}} q_{\hat{h}j} x_j > b_{\hat{h}} \\ \Leftrightarrow & \sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{\hat{h}j} \frac{v_j}{\lambda} + \sum_{l \in \mathcal{S}} q_{\hat{h}l} \frac{v_l}{\lambda} > b_{\hat{h}} \\ \Leftrightarrow & \sum_{j \in \mathbf{N} \setminus \mathcal{S}} q_{\hat{h}j} v_j + \sum_{l \in \mathcal{S}} q_{\hat{h}l} v_l > \lambda b_{\hat{h}}. \end{aligned}$$

This contradicts the feasibility of (v, λ) as it would violate the constraint (5.23) for index $\hat{h} \in \mathbf{H}$. Therefore, there exists no such solution (v, λ) . $\square$

Consequently, when the solution to CP3 induces an integer descent direction d , the ensuing solution $x^0 + d$ is always feasible for problem $\mathbb{P}$. Unfortunately, CP3 can also find solutions yielding a fractional direction d . In this case, we propose to search for an integer descent

direction d , called a *subdirection* of d , as follows. Let $S_d^+ = \{j \in N \mid d_j > 0\}$ and $S_d^- = \{j \in N \mid d_j < 0\}$ be the indices of the columns that should enter and exit the current solution $\mathcal{S}$ according to direction d , respectively.

Definition 8. A direction d^I is said to be a subdirection of direction d if

$$d_j^I = 0, \quad \forall j \in N \setminus (S_d^+ \cup S_d^-), \quad d_j^I \geq 0, \quad \forall j \in S_d^+, \quad d_j^I \leq 0, \quad \forall j \in S_d^-.$$

Given that the sets S_d^+ and S_d^- are typically small, we solve a mixed-integer program (MIP) to find an integer descent subdirection d^I of direction d . This MIP aims at replacing some columns in the current solution by others, ensuring that the resulting solution is feasible. For each $h \in \mathbf{H}$, let $\hat{s}_h = b_h - \sum_{j \in \mathcal{S}} q_{hj}$ be the slack left in the side constraint h by the current solution $\mathcal{S}$. Furthermore, let y_j^+ , $j \in S_d^+$, and y_j^- , $j \in S_d^-$, be binary variables associated with the columns in S_d^+ and S_d^- . Variable y_j^+ (resp., y_j^-) takes value 1 if the column indexed by j enters (resp., leaves) the current solution. The proposed MIP is:

$$z^{SD} = \min_{y^+, y^-} \sum_{j \in S_d^+} c_j y_j^+ - \sum_{j \in S_d^-} c_j y_j^- \quad (5.29)$$

$$\text{s.t.:} \quad \sum_{j \in S_d^+} a_{ij} y_j^+ - \sum_{j \in S_d^-} a_{ij} y_j^- = 0, \quad \forall i \in \mathbf{T} \quad (5.30)$$

$$\sum_{j \in S_d^+} q_{hj} y_j^+ - \sum_{j \in S_d^-} q_{hj} y_j^- \leq \hat{s}_h, \quad \forall h \in \mathbf{H} \quad (5.31)$$

$$y_j^+ \in \{0, 1\}, \quad \forall j \in S_d^+, \quad y_j^- \in \{0, 1\}, \quad \forall j \in S_d^-. \quad (5.32)$$

Objective function (5.29) aims at minimizing the cost of replacing the selected columns in S_d^- by those selected in S_d^+ . Constraints (5.30) impose that the entering columns cover exactly the same tasks as the exiting columns. Finally, constraints (5.31) ensure that no side constraints are violated by this change. Note that the number of constraints can be reduced by considering only those for which there exists at least one column in $S_d^+ \cup S_d^-$ with a non-zero coefficient in this constraint.

An integer descent subdirection is found if $z^{SD} < 0$. This subdirection is given by

$$d_j^I = \begin{cases} y_j^+ & \text{if } j \in S_d^+ \\ -y_j^- & \text{if } j \in S_d^- \\ 0 & \text{otherwise.} \end{cases}$$

5.4.3 Possible RP-CP combinations

In this subsection, we discuss possible RP-CP combinations that can be used in ISUD to solve the SPPSC.

- **RP2-CP2:** This combination consists of handling the side constraints only in the RP. Since this problem involves subsets of constraints and columns, it can be easily solved using a commercial solver. However, the CP may find descent directions that do not point to a feasible region of $\text{conv}(\mathcal{F}_{\text{SPPSC}})$, i.e., that points to a region where side constraints are violated. Therefore, to find an improved integer solution, it is often necessary to define a large neighborhood when zooming around these directions and the RP may, thus, consume more computational time.
- **RP1-CP3:** In this combination, the side constraints are only handled in the CP, ensuring that the descent directions found lead to a feasible region. However, if the directions found are fractional, the zooming procedure cannot be used since the RP does not consider the side constraints and, therefore, the solution process may terminate before reaching an optimal solution.
- **RP2-CP3:** With this combination, the advantages of handling the side constraints both in the RP and in the CP are exploited. In addition, the zooming procedure can be used in the RP to search for integer solutions in the neighborhood of any fractional direction found by the CP.

Given that the second combination offers no recourse when fractional directions are found, it is not considered in the following. We only compare the first and third combinations (i.e., RP2-CP2 and RP2-CP3) which rely on the zooming procedure to search for integer directions around fractional ones.

5.5 Improved integral column generation (I²CG)

In this section, we describe the proposed I²CG algorithm, an improved version of ICG [40], that can handle the side constraints of the SPPSC. Like ICG, it is based on a three-level decomposition. The first two levels correspond to the CG RMP which is decomposed into a RP and a CP. The third level consists of the CG PS which is, in our case, a shortest path problem with resource constraints that is solved by a labeling algorithm.

A pseudo-code for I²CG is presented in Algorithm 3. It starts with an initial set of columns $\tilde{\mathbf{N}}$ (possibly artificial ones), an integer solution x^0 (also denoted $\mathcal{S}^0$) of cost z^0 , and a corresponding dual solution π^0 . Then, it tries to improve this initial solution by CG. At each CG iteration, the PS is solved in Step 3 and the RMP is solved by ISUD in Steps 7–16 and, if

Algorithm 3: I²CG

input : initial columns $\tilde{\mathbf{N}}$, solution x^0 ($\mathcal{S}^0$), dual solution π^0 , and cost z^0
output : best integer solution found x^*
parameter : CPversion, maxZoom, kZoom, minImprPerc, maxSuccFail,
 $\mathbf{K}^{CP} = \{k_1, \dots, k_\eta\}$

```

1   $r \leftarrow 0$ ;  $nbSuccFail \leftarrow 0$ ;  $x^* \leftarrow x^0$ ;  $\mathcal{S} \leftarrow \mathcal{S}^0$ ;
2  do
    Column generation
3   $\mathbf{J} \leftarrow \text{solve PS}(\pi^r)$ ;  $r \leftarrow r + 1$ ;
4  if  $\mathbf{J} \neq \emptyset$  then
5       $\tilde{\mathbf{N}} \leftarrow \tilde{\mathbf{N}} \cup \mathbf{J}$ ;
        RP-CP loop
6       $k \leftarrow k_1$ ;
7      do
8           $z^r \leftarrow \text{solve RP2}(\mathcal{S})$ ;
9          do
10              $(d, \pi^r) \leftarrow \text{solve CPversion}(\tilde{\mathbf{N}}_{\mathcal{S}}^k)$ ;
11             if  $d$  is integer or contains an integer subdirection then
12                 update  $x^*$  and  $\mathcal{S}$ ;
13                 break;
14              $k \leftarrow \text{nextPhase}(k, \mathbf{K}^{CP})$ ;
15             while  $k \neq 0$ ;
16         while  $d$  is integer or contains an integer subdirection;
        Zooming
17         if  $d$  is fractional then
18              $\mathcal{S}' \leftarrow \mathcal{S}$ ;  $d^1 \leftarrow d$ ;
19             for  $\ell = 1$  to maxZoom do
20                  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \text{supp}(d^\ell)$ ;
21                  $z^r \leftarrow \text{solve RP2}(\mathcal{S}')$ ;
22                 if improved integer solution found then
23                     update  $x^*$  and  $\mathcal{S}$ ;
24                     break;
25                 if  $\ell < \text{maxZoom}$  then
26                      $d^{\ell+1} \leftarrow \text{solve CPversion}(\tilde{\mathbf{N}}_{\mathcal{S}'}^{k_{\text{Zoom}}})$ ;
27                     if  $d^{\ell+1} = d^\ell$  then
28                         break;
        Control
29         if  $\frac{(z^{r-1} - z^r)}{z^{r-1}} < \text{minImprPerc}$  then
30              $nbSuccFail \leftarrow nbSuccFail + 1$ ;
31         else
32              $nbSuccFail \leftarrow 0$ ;
33 while  $\mathbf{J} \neq \emptyset$  and  $nbSuccFail < \text{maxSuccFail}$ ;

```

needed, by invoking the zooming procedure in Steps 17–28. Finally, Steps 29–32 count the number of successive CG iterations where the progress made in the objective value is deemed insufficient. This counter is used in the algorithm stopping criterion tested in Step 33. Note that, in this pseudo-code, we use $\text{RP2}(\sigma)$ to specify that RP2 is built with respect to solution σ ($= \mathcal{S}$ or $\mathcal{S}'$), and $\text{CPversion}(\mathbf{L})$ to identify the CP of version CPversion ($= \text{CP2}$ or CP3) defined by replacing set $\mathbf{N}$ with set $\mathbf{L}$. Below, we discuss the main parts of this pseudo-code in separate subsections.

5.5.1 Solving the pricing subproblem

In Step 3, the PS (which may be separated in multiple PSs) is solved by, e.g., a labeling algorithm to find a set $\mathbf{J}$ of negative reduced cost columns, where the reduced cost of column j is computed as $\bar{c}_j = c_j - \sum_{t \in \mathbf{T}} \pi_t a_{tj} - \sum_{h \in \mathbf{H}} \pi_h q_{hj}$. If $\mathbf{J}$ is empty, the algorithm stops. Otherwise, the columns in $\mathbf{J}$ are added to the current RMP column pool $\tilde{\mathbf{N}}$ in Step 5. Note that a multi-phase strategy that initially restricts the PS solution space and gradually enlarges it when no “good” negative reduced columns can be found is also applied when solving the PS. Details about this strategy are provided at the end of Section 5.5.2.

For both versions of the CP, the dual solution $\pi^r = ((\pi_t^r)_{t \in \mathbf{T}}, (\pi_h^r)_{h \in \mathbf{H}})$ used in the PS at iteration r is either provided by the initial dual solution π^0 (when $r = 0$) or by optimal dual values obtained in the last solution of RP2 in Step 8 or the CP in Step 10. More precisely, if CP2 is used in I^2CG , then the duals $\pi_t^r, \forall t \in \mathbf{T}$, and $\pi_h^r, \forall h \in \mathbf{H}$, are set equal to those of constraints (5.14) and (5.19), respectively. This choice of dual solution is based on empirical results and offers no guarantee of algorithmic convergence. On the other hand, when I^2CG relies on CP3 , both sets of dual values are provided by CP3 , namely, from constraints (5.22) and (5.23). In this case, the following proposition can be proven.

Proposition 6. *Let $\mathcal{S}$ be an integer solution with cost $c_{\mathcal{S}} = \sum_{j \in \mathcal{S}} c_j$. Let $\mathcal{S}^*$ be an improved solution ($\sum_{j \in \mathcal{S}^*} c_j < c_{\mathcal{S}}$). At least one variable $x_j, j \in \mathcal{S}^*$, has a negative reduced cost with respect to the dual solution $\pi \in \mathbb{R}^{|\mathbf{T}|+|\mathbf{H}|}$ provided by (5.22) and (5.23).*

Proof. After solving CP3, the reduced cost of λ is always equal to 0, which implies

$$c_{\mathcal{S}} = \sum_{i \in \mathbf{T}} \pi_i + \sum_{h \in \mathbf{H}} b_h \pi_h. \quad (5.33)$$

Furthermore, given that $\pi_h \leq 0$ and $\sum_{j \in \mathcal{S}^*} q_{hj} \leq b_h$ for all $h \in \mathbf{H}$, the following relation holds:

$$\sum_{h \in \mathbf{H}} \sum_{j \in \mathcal{S}^*} q_{hj} \pi_h \geq \sum_{h \in \mathbf{H}} b_h \pi_h. \quad (5.34)$$

Now, let us assume that all variables x_j , $j \in \mathcal{S}^*$, have a non-negative reduced cost, i.e.,

$$\bar{c}_j = c_j - \sum_{t \in \mathbf{T}} \pi_t a_{tj} - \sum_{h \in \mathbf{H}} \pi_h q_{hj} \geq 0, \quad \forall j \in \mathcal{S}^*. \quad (5.35)$$

These inequalities yield

$$\begin{aligned} \sum_{j \in \mathcal{S}^*} (c_j - \sum_{i \in \mathbf{T}} a_{ij} \pi_i - \sum_{h \in \mathbf{H}} q_{hj} \pi_h) \geq 0 &\Rightarrow \sum_{j \in \mathcal{S}^*} c_j \geq \sum_{i \in \mathbf{T}} \pi_i + \sum_{j \in \mathcal{S}^*} \sum_{h \in \mathbf{H}} q_{hj} \pi_h \\ &\Rightarrow \sum_{j \in \mathcal{S}^*} c_j \geq \sum_{i \in \mathbf{T}} \pi_i + \sum_{h \in \mathbf{H}} b_h \pi_h \quad (\text{from (5.34)}) \\ &\Rightarrow \sum_{j \in \mathcal{S}^*} c_j \geq c_{\mathcal{S}} \quad (\text{from (5.33)}). \end{aligned}$$

This contradicts the hypothesis that $\mathcal{S}^*$ is an improved solution. Therefore, the above assumption is false and at least one variable x_j , $j \in \mathcal{S}^*$, has a negative reduced cost. $\square$

The following corollary, which provides an optimality certificate, directly ensues from this proposition.

Corollary 1. *If $z^{CP3} = 0$ and no negative reduced cost columns are generated when solving the PS using the dual solution $\pi \in \mathbb{R}^{|\mathbf{T}|+|\mathbf{H}|}$ provided by (5.22) and (5.23), then the current solution $\mathcal{S}$ is optimal for $\mathbb{P}$.*

If $z^{CP3} < 0$ and $\mathcal{S}$ is not optimal (i.e., there exists an improved solution $\mathcal{S}^*$), then there is no guarantee that the PS can generate a negative reduced cost column. Indeed, even if Proposition 6 stipulates that there exists at least one variable x_j , $j \in \mathcal{S}^*$, that has a negative reduced cost, all negative reduced cost variables may belong to the set $\tilde{\mathbf{N}}$ of already generated columns. In this case, nonnegative reduced cost columns have to be generated by the PS and added to $\tilde{\mathbf{N}}$ to find the corresponding descent direction. Identifying these columns does not seem feasible. Alternatively, to ensure the exactness of the algorithm, one can restrict CP3 by including the disjunctive constraints (5.12) and generating negative reduced cost columns when branching to satisfy them. Given that we do not necessarily look for optimal solutions in our computational experiments, this option has not been implemented. Furthermore, our computational results show that this situation is not frequent, at least not before being very close to optimality.

5.5.2 RP-CP loop

In the RP-CP loop (Steps 7-16), ISUD is applied to solve the RMP with the goal of finding the best integer solution given the available set of columns $\tilde{\mathbf{N}}$. This loop stops, however,

when no integer direction or subdirection is found by the CP. Note that the CP version used is passed to the algorithm as a parameter (*CPversion*). In the remainder of this paper, we denote by I^2CG_2 and I^2CG_3 the I^2CG algorithm based on $CPversion = CP2$ and $CPversion = CP3$, respectively.

Theoretically, in a given CG iteration r , an optimal solution to the RMP can be found by solving a single CP3 because this solution is adjacent to the aggregated solution (see Theorem 1). However, in practice, several iterations in the RP-CP loop may be necessary to reach this solution because of the objective function of CP3, which minimizes an “average” reduced cost. Indeed, one can observe that the cost of a solution (v, λ) to CP3 yielding an integer descent direction d is equal to $\lambda c^\top d$, where d and λ are defined by (5.27)–(5.28). In particular, if the weights α_j and β_l in (5.24) are all equal to 1, then $\lambda = 1/n^+(v)$, where $n^+(v)$ is equal to the number of v_j variables with a positive value, i.e., the number of columns in the solution $x^0 + d$. In this case, the cost of (v, λ) is equal to the average reduced cost per column in this solution.

For the SPP, Zaghroui *et al.* [8] have shown that there exists weights α_j^* and β_l^* such that the direction induced by the corresponding solution of CP2 leads to an optimal solution of the problem. Such result can be generalized to CP3 and the SPPSC RMP. Unfortunately, there is no method to identify such weights a priori.

For our tests, we used the following weights: $\alpha_j = \delta^j$, $j \in \mathbf{N} \setminus \mathbf{C}_S$, $\alpha_j = 2$, $j \in \mathbf{C}_S \setminus \mathcal{S}$, and $\beta_l = 1$, $l \in \mathcal{S}$, where δ^j is the degree of incompatibility of column A_j . This incompatibility degree can be mathematically defined as the “distance” between column A_j and the vector subspace generated by the columns in $\mathcal{S}$ (omitting the side constraint coefficients); a formal definition is given below. Using these weights, CP3 aims at minimizing the average reduced cost per degree of incompatibility in the columns of the resulting solution $x^0 + d$. In fact, empirical results show that improved integer solutions are often composed of columns with a small degree of incompatibility. Thus, this objective function favors finding them.

The degree of incompatibility of a column A_j , $j \in \mathbf{N}$, is defined according to the current solution $\mathcal{S}$ and a so-called compatibility matrix M .

Definition 9. A matrix M is a compatibility matrix if and only if $\|MA_j\|_1 = 0$ for every compatible column A_j , $j \in C_S$.

Definition 10. The incompatibility degree of a column A_j , $j \in \mathbf{N}$, is $\delta^j = \|MA_j\|_1$.

Note that $\delta^j = 0$ for all compatible columns A_j , $j \in C_S$.

In our algorithm, we use the compatibility matrix M_2 of Bouarab *et al.* [20] which is designed for vehicle routing and crew scheduling applications. With this matrix, the degree of

incompatibility of a column A_j indicates the number of times that the task subsets of the current solution $\mathcal{S}$ must be divided to ensure that A_j becomes compatible. For example, if two columns in $\mathcal{S}$ cover the task subsets $\{1, 2, 3, 4\}$ and $\{5, 6\}$, then $\delta^j = 2$ for an incompatible column A_j covering the task subset $\{4, 5\}$ because, to make A_j compatible, both subsets $\{1, 2, 3, 4\}$ and $\{5, 6\}$ must be divided to yield the subsets $\{1, 2, 3\}$, $\{4\}$, $\{5\}$, and $\{6\}$.

To accelerate Algorithm 3 and further favor the identification of improved integer solutions made up of columns with a small degree of incompatibility, a multi-phase strategy similar to those developed by El Hallaoui *et al.* [17] for the DCA algorithm and adapted by Zaghrouti *et al.* [6] for the ISUD algorithm is implemented in Steps 9–15. Let $\mathbf{K}^{CP} = \{k_1, k_2, \dots, k_\eta\}$ be an ordered set of phases, where $k_i, i = 1, 2, \dots, \eta$, is a maximum degree of incompatibility, $k_i \geq 1$ and $k_i < k_{i+1}$ for all $i = 1, 2, \dots, \eta - 1$. For example, $\mathbf{K}^{CP}$ may be equal to $\{1, 2, 3, 5, 8\}$. In a phase $k \in \mathbf{K}^{CP}$, the CP solved in Step 10 is restricted to a subset $\tilde{\mathbf{N}}_{\mathcal{S}}^k$ of the generated columns, namely, those with an incompatibility degree less than k with respect to solution $\mathcal{S}$. Therefore, every subset $\tilde{\mathbf{N}}^k, k \in \mathbf{K}^{CP}$, contains all compatible columns and subset $\tilde{\mathbf{N}}^{k_\eta} = \tilde{\mathbf{N}}$ if k_η is sufficiently large.

Each CG iteration starts in phase $k = k_1$. Then, after solving the RP in Step 8, the search for a descent direction starts in the current phase k . If the computed direction d in Step 10 is integer or contains an integer subdirection, then the best found solution is updated. Otherwise, in Step 14, function *nextPhase* identifies the next phase to explore in set $\mathbf{K}^{CP}$ if $k < k_\eta$ and returns $k = 0$ otherwise. In the former case, the CP is solved again. In the latter, the algorithm continues with the zooming procedure.

As mentioned in the previous section, we also implemented a multi-phase strategy for the PS that is independent from the one described above, using a set of phases $\mathbf{K}^{PS}$. The objective of this strategy is to generate, at the beginning of the algorithm, columns that are slightly incompatible with the current best integer solution. In phase $k \in \mathbf{K}^{PS}$, the PS (a shortest path problem with resource constraints) involves an additional resource constraint that limits the incompatibility degree of the generated columns to be less than or equal to k (see El Hallaoui *et al.* [18] for further details). The algorithm starts in the first phase in $\mathbf{K}^{PS}$. It moves to the next phase either when no column is generated in Step 3 (in which case, the PS is immediately solved again) or when a failure is identified in Step 29.

5.5.3 Zooming

When the last direction d found by the CP in Step 10 is fractional, the *zooming* procedure (Steps 18–28) is called to look for an improved solution in the neighborhood of this direction. The first neighborhood is created by adding the column indices j such that $d_j > 0$ to the

index set $\mathcal{S}$ to form an augmented index set $\mathcal{S}'$ (Step 20). Then, RP2 defined with the variables compatible with $\mathcal{S}'$ is solved in Step 21 with the hope of finding an improved integer solution. If this is the case, the best solution found is updated and the zooming procedure stops. Otherwise, the neighborhood is enlarged by solving the CP (Step 26) to find a new descent direction and increase set $\mathcal{S}'$, before solving again the RP. In total, at most $maxZoom$ neighborhoods are explored before halting the zooming procedure. To keep the size of the CP relatively small and favor finding improved solutions composed of columns with a small degree of incompatibility, the CP is defined for the set $\tilde{\mathbf{N}}_{\mathcal{S}'}^{kZoom}$ of columns, i.e., those having a degree of incompatibility with respect to $\mathcal{S}'$ less than or equal to the value of parameter $kZoom$. Note that, because set $\mathcal{S}'$ increases at each iteration, the size of $\tilde{\mathbf{N}}_{\mathcal{S}'}^{kZoom}$ increases at each iteration and the degree of incompatibility of its columns with respect to the current solution $\mathcal{S}$ can be larger than $kZoom$. Nevertheless, the direction $d^{\ell+1}$ found at iteration ℓ may be the same as d^ℓ , the one obtained in the previous iteration. In this case (Step 27), set $\mathcal{S}'$ cannot increase and the zooming procedure stops.

5.5.4 Control and stopping condition

Once the current CG iteration is completed, i.e., after solving the PS (pricing subproblem) and the RMP through the RP-CP loop and, possibly, the zooming procedure, the improvement in the cost of the best solution found in this iteration is evaluated in Step 29 of Algorithm 3. If this improvement in percentage does not exceed the value of parameter $minImprPerc$, then this CG iteration is deemed a failure and the number of successive iterations with a failure is increased by 1 (Step 30). Otherwise, it is qualified as a success and the number of successive iterations with a failure is reset to 0 in Step 32.

The main while loop stops in Step 33 when the PS cannot generate negative reduced cost columns or when a failure occurs in $maxSuccFail$ successive iterations.

5.6 Computational results

In this section, we present the results of the computational tests that we performed on real-life instances of the CPP with up to 1740 flights. Each instance (derived from the datasets of Kasirzadeh *et al.* [39]) concerns a single aircraft type with a horizon of one week and three bases. To each instance, we added 21 (7 days times 3 bases) side constraints to limit the number of pilots available per base and day. For each constraint $h \in \mathbf{H}$ and pairing $j \in \mathbf{N}$, coefficient q_{hj} is equal to 1 if j and h are associated with the same base and j covers a flight during the day associated with h . Furthermore, the right-hand side b_h of constraint $h \in \mathbf{H}$

corresponds to the number of pilots available in the base associated with h . For our tests, set $\mathbf{N}$ is initialized with artificial columns, namely, one for each task $i \in \mathbf{T}$ with no contribution to the side constraints and a large cost M . These columns form the initial solution $\mathcal{S}^0$ which has a cost $z^0 = M|\mathbf{T}|$. The dual solution π^0 is thus given by $\pi_i^0 = M, \forall i \in \mathbf{T}$, and $\pi^0 = 0, \forall h \in \mathbf{H}$.

In our experiments, we tested both versions of I²CG (I²CG₂ and I²CG₃), a diving heuristic (DH), and a restricted master heuristic (RMH) (see Joncour *et al.* [37]). DH is a BP heuristic which explores a single branch of the search tree in a depth-first fashion without any backtracking. After solving each linear relaxation by CG, it fixes the pairing variable with the highest fractional value to 1 until finding an integer solution. RMH starts by solving the linear relaxation of the SPPSC by CG and converts the last RMP into a MIP which is then solved by a commercial solver. It should be noted that DH and RMH do not guarantee to find an integer solution. In fact, as no backtracking is allowed in DH, the algorithm may end up with an infeasible RMP after fixing a certain number of variables to 1. For RMH, the subset of columns generated for solving the linear relaxation of the problem may simply not contain a feasible integer solution. For both I²CG₂ and I²CG₃, the following parameter values were used: $minImprPerc = 0.0025$, $maxSuccFail = 9$, $kZoom = 5$, $\mathbf{K}^{CP} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathbf{K}^{PS} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Furthermore, we set $maxZoom = 4$ for I²CG₂ and $maxZoom = 2$ for I²CG₃. All these values were determined based on preliminary test results.

All our tests were run on a Linux machine equipped with an Intel i7 processor clocked at 3.4 GHz. In all algorithms, the SPs were solved using dynamic programming implemented using the Boost library (version 1.54). All linear RMPs and MIPs were solved by CPLEX (version 12.6).

This section is divided in three parts. First, we compare the performance of the algorithms I²CG₂, I²CG₃, DH, and RMH. Second, we present detailed results to provide insights on the performance of I²CG₃, compared to DH and RMH. Finally, we study the impact of the side constraint right-hand side values on the computational times of all algorithms.

5.6.1 Performance tests

In Table 5.2, we report computational results that allow to compare the performance of the algorithms I²CG₂, I²CG₃, DH, and RMH. Each instance is identified by an Id (*Id*) and the number of tasks it involves (*Tasks*). Then, for each algorithm, we read from left to right: the total computational time in seconds (*Time*), the optimality gap in percentage between the cost of the best solution found and the linear relaxation optimal value (*Gap*), the number of

column generation iterations (*Itr.*), the total number of integer solutions found (*IntS*), the percentage of saturated side constraints in the best solution found (*Sc.*), and the total number of generated columns (*Cols*). For DH, the number of integer solutions is not indicated, as it is always equal to 1. It should be noted that the linear relaxation optimal values were only computed to report the optimality gaps, so the time required to calculate them is not included in the total time of the I²CG algorithms. In the last row of the table, we report the average of each table column. For each instance, the least computational time and the least gap among all algorithms are highlighted in bold.

Table 5.2 Results of I²CG₂, I²CG₃, DH and RMH.

<i>Instance</i>		<i>I²CG₂</i>						<i>I²CG₃</i>						<i>DH</i>					<i>RMH</i>					
<i>Id</i>	<i>Tasks</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>
727	239	9	0.36	23	73	33	23413	8	0.35	20	66	42	19848	9	0.35	59	33	9474	5	0.42	21	5	33	8674
09	348	13	0.18	14	56	9	24408	13	0.16	19	56	9	28438	5	0.36	26	9	10695	4	0.20	17	5	9	10645
94	424	44	0.21	19	93	66	50823	36	0.19	18	101	61	57028	38	0.45	71	66	19615	15	0.26	23	5	71	18184
95	1255	2411	0.17	24	455	42	457784	1669	0.17	23	489	42	419540	8052	0.22	903	52	171415	7640	1.19	53	30	52	103250
757	1290	915	0.01	17	324	4	326725	855	0.01	17	402	4	337999	3479	0.03	756	14	105320	1163	0.16	75	8	19	62791
319	1293	1482	0.17	20	488	42	277853	1542	0.11	23	462	47	283315	4514	0.24	1170	42	133403	7406	2.14	89	19	76	77687
320	1740	2034	0.06	24	438	38	456921	1874	0.05	17	565	33	369921	8126	0.08	1317	38	151183	1144	0.15	103	13	42	91577
Avg.		987	0.17	20	275	33	231132	857	0.15	20	306	34	216584	3460	0.24	614	36	85872	2482	0.64	54	12	43	53258

From these results, we observe that I²CG₂ finds integer solutions with a slightly higher average optimality gap than those obtained with I²CG₃. This is not surprising because I²CG₂ does not handle the side constraints in CP2 and the descent directions found by CP2 can lead to infeasible regions. So, to increase the probability of finding integer solutions in the neighborhood of these directions via the zooming procedure, I²CG₂ must search in larger neighborhoods than I²CG₃. This justifies: i) setting *maxZoom* = 4 for I²CG₂ instead of *maxZoom* = 2 for I²CG₃; ii) larger computational times for I²CG₂ except for one instance.

We can also observe that I²CG₃ generally finds better solutions faster than other methods. Indeed, the solution found by I²CG₃ has an optimality gap that can be 3 times smaller than the gap of the solution found by DH. In addition, I²CG₃ reduces the DH time by a factor of 3 for instance 319 and of 4 for instances 95, 757 and 320. On average, I²CG₃ is 4 times faster than DH, 2.8 times faster than RMH, and improves the optimality gap by a factor of 1.6 for DH and of 4 for RMH. We also notice that, on average, I²CG₃ generates 2.5 times more columns than DH. This can be explained by the extra resource that computes the incompatibility degree in the I²CG₃ SPs (see Section 5.5.2) which reduces label dominance and also by the absence of dominance at the network sink nodes. This choice of generating a large number of columns was justified in Tahir *et al.* [40], where the authors showed that this strategy is, contrary to DH, beneficial to ICG. In fact, the large number of generated columns can be efficiently managed by I²CG in the RMP using the multi-phase strategy (see Section 5.5.2). This speed up can also be explained by the number of iterations required by

I^2CG_3 (20 on average), which is negligible compared to that of DH (614 on average).

Another remarkable characteristic of the I^2CG_3 algorithm is the large number of integer solutions found throughout the solution process. Indeed, I^2CG_3 finds 33 integer solutions per iteration for the largest instance 320 and 15 integer solutions per iteration on average over all instances. This corresponds to an improved solution at every 3 seconds on average. This characteristic of I^2CG_3 is highly desirable in the industry where the solution can sometimes be stopped prematurely because of a deadline for producing the crew schedules.

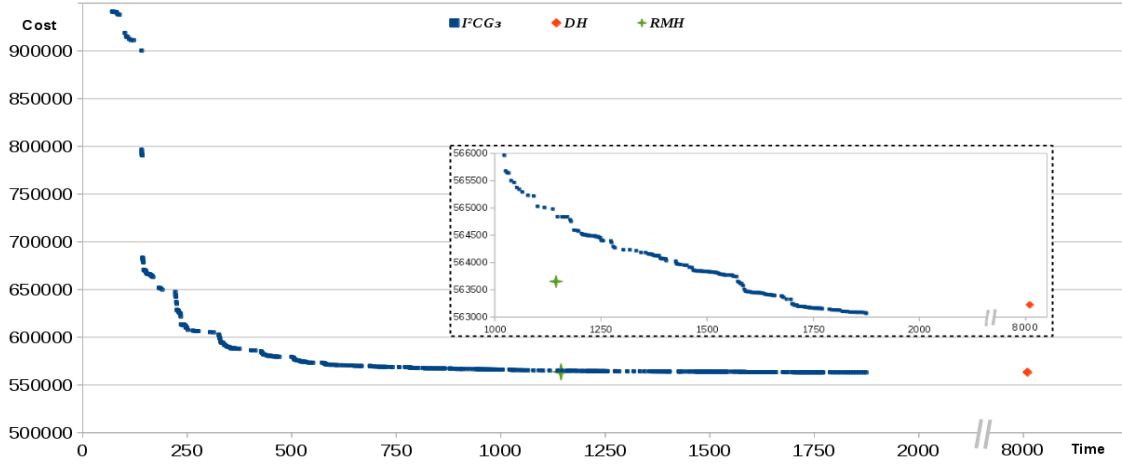


Figure 5.2 Cost of the current solution in function of the computational time – Instance 320.

Figure 5.2 illustrates the evolution of the cost of the current solution as a function of the computational time for instance 320 (a similar behavior is observed for the other instances). It includes an enlargement of the function between times 1000 and 8100 seconds. Each blue spot indicates an integer solution found by I^2CG_3 . The red diamond and the green star represent the unique solution found by DH and the best one computed by RMH. This figure shows the fast improvement of the current solution at the beginning of the I^2CG_3 solution process. Like other column generation methods, a tailing off occurs, i.e., the current cost slowly improves in the second half of this process. We also note that, for I^2CG_3 , integer solutions are found regularly, which means that I^2CG_3 is not prone to degeneracy.

5.6.2 Some insights

In Tables 5.3 and 5.4, we detail the I^2CG_3 , DH and RMH results reported in Table 5.2. The objective is to analyse the behavior of different components of the three algorithms. In these tables, we give for each algorithm the time consumed by each algorithm component in seconds (*Time*), the number of times that the PSs were solved (*Itr.*), i.e., the number of CG iterations, and the average number of generated columns per iteration (*Av.GCol*). For I^2CG_3 ,

we also report the number of integer solutions found by RP2 ($IntS$), the average number of constraints in RP2 ($Av.Cst.$), the average number of non-zero coefficients ($Av.Nz.$) in the RP2 constraint matrix, the number of solved RP2s ($No.$), the number of integer directions found by CP3 ($IntD$), the number of integer subdirections ($IntSD$), and the average number of columns in CP3 ($Av.Cols.$). For RMH, we also provide the number of non-zero coefficients in the MIP ($Nz.$). Averages of these results are given in the last row of each table. Recall that I^2CG_3 can find an integer solution by solving either RP2 or CP3.

Table 5.3 Detailed results of I^2CG_3 .

Instance		I^2CG_3												
		RP2					CP3					PS		
<i>Id</i>	<i>Tasks</i>	<i>Time</i>	<i>IntS</i>	<i>Av.Cst.</i>	<i>Av.Nz.</i>	<i>No.</i>	<i>Time</i>	<i>intD</i>	<i>intSD</i>	<i>Av.Col.</i>	<i>No.</i>	<i>Time</i>	<i>Itr.</i>	<i>Av.GCol.</i>
727	239	1	25	115	2997	90	3	24	17	1389	223	2	20	992
09	348	1	9	179	3967	88	7	17	30	2696	244	3	19	1497
94	424	6	28	206	6146	111	18	33	42	5190	235	7	18	3168
95	1255	335	112	546	18096	302	983	43	334	29344	456	270	23	18241
757	1290	8	46	689	10472	206	691	3	353	33096	310	129	17	19882
319	1293	423	130	539	13497	258	941	9	323	25998	399	127	23	12318
320	1740	292	203	806	15222	256	1337	21	341	37580	359	178	17	21760
Avg.		152	79	440	10056	187	568	21	205	19327	318	102	19	11123

Table 5.4 Detailed results of DH and RMH.

Instance		DH				RMH					
		RMP	PS			RMP	MIP		PS		
<i>Id</i>	<i>Tasks</i>	<i>Time</i>	<i>Time</i>	<i>Itr.</i>	<i>Av.GCol.</i>	<i>Time</i>	<i>Time</i>	<i>Nz.</i>	<i>Time</i>	<i>Itr.</i>	<i>Av.GCol.</i>
727	239	2	7	59	160	1	1	81336	2	21	413
09	348	1	4	26	411	1	1	102728	2	17	626
94	424	5	32	71	276	2	3	180240	8	23	790
95	1255	430	7605	903	189	60	7003	1112762	576	53	1948
757	1290	299	3169	756	139	74	771	533957	318	75	837
319	1293	468	4024	1170	114	97	7001	748058	307	89	872
320	1740	706	7401	1317	114	194	366	842719	583	103	889
Avg.		273	3177	614	200	61	2163	514542	256	54	910

From Table 5.3, we observe that CP3 consumes on average more than 70% of the total computational time of I^2CG_3 , with an average of 2 seconds per CP3 solved. It finds more than 70% of the integer solutions identified by the algorithm. This is not surprising as the descent directions found by CP3 can reach any improved integer solution of the SPPSC according to Theorem 1. Integer solutions obtained from CP3 are derived from an integer descent direction in 10% of the times and from an integer descent subdirection in 90% of the times. We also observe that RP2 requires less than 1 second per resolution. This can be explained by its average number of constraints that does not exceed one-half of the number of constraints in the original problem. Finally, note that CP3 contains an average number of columns that is not very large, which makes it relatively easy to solve at each iteration.

From Table 5.4, we make the following observations. In DH and RMH, the average time consumed by the PS per CG iteration is less than that used in I^2CG_3 for the largest instances.

This is mainly due to the incompatibility resource added in the PS of I^2CG_3 . Nevertheless, the PS in DH takes, on average, more than 91% of the total time, while it requires only 13% of it in I^2GC_3 . This difference is mainly explained by the number of calls to the PS and the fact that the RMP is a linear program in DH while it is an integer program solved by ISUD in I^2GC_3 . For RMH, we see that the MIP consumes on average more than 80% of the total time. The average size of the solved MIPs is very large compared to the average size of the RPs solved in I^2GC_3 (see Table 5.2). Finally, observe that the average number of columns generated per iteration in DH is much less than in RMH. This is explained by the large number of columns generated in the first CG iterations and that RMH stops generating columns after solving the linear relaxation.

To conclude this section, we briefly discuss the convergence of the minimum reduced cost in the I^2CG_3 algorithm. For instance 320, Figure 5.3 depicts the minimum reduced cost computed at each CG iteration (by solving the PSs) for I^2CG_3 (left) and DH (right). These results show that the dual variables used to generate columns in I^2CG_3 are more stable than those used by DH. In fact, every time that variables are fixed in DH, some of the dual variables are highly perturbed, yielding the generation of columns that may have little chance to be part of the final integer solution. Such behavior is not observed for I^2CG_3 , which may explain its rapid convergence.

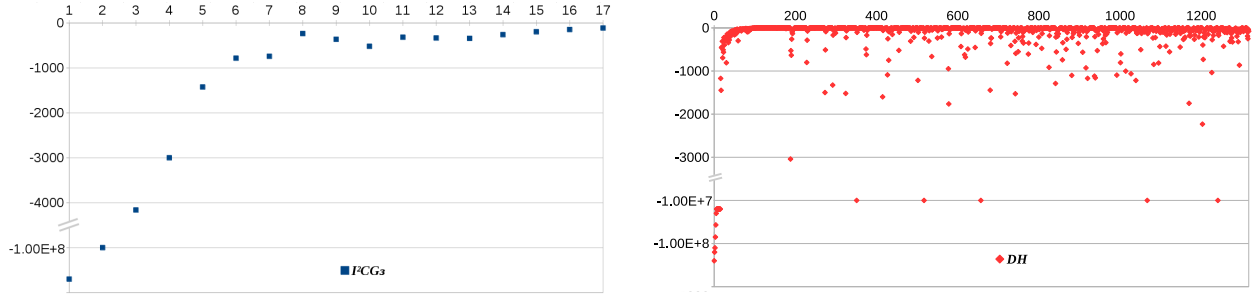


Figure 5.3 Minimum reduced cost at each CG iteration for I^2CG_3 and DH – Instance 320.

5.6.3 Sensitivity of I^2CG_3 to right-hand side values

To analyze the sensitivity of the I^2CG_3 algorithm to the tightness level of the side constraints and validate that it still outperforms the other algorithms for various levels, we conducted another series of experiments with additional instances derived from the instances used for our main tests. For each of them, we created four other instances: one with larger supplementary constraint right-hand side values (i.e., with less tight constraints) and three with increasingly smaller values. The tightness levels are numbered from 1 (loosest) to 5 (tightest), with level 2 corresponding to the original level. The average results of these experiments are reported

in Table 5.5, where column *Level* indicates the tightness level of the instance and the other columns have the same meaning as in Table 5.2. Note that, for level 2, we report the same average results as in Table 5.2. Instance-by-instance results can be found in Appendix B.1.

Table 5.5 Sensitivity analysis average results.

<i>Level</i>	<i>I²CG₂</i>						<i>I²CG₃</i>						<i>DH</i>					<i>RMH</i>					
	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>Sc.</i>	<i>Cols</i>	<i>Time</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Sc.</i>	<i>Cols</i>
1	969	0.14	20	274	15	215534	737	0.12	19	294	17	215109	3605	0.23	651	15	83596	2374	0.39	54	13	21	50950
2	987	0.17	20	275	33	231132	857	0.15	20	306	34	216584	3460	0.25	615	36	85872	2482	0.65	54	12	43	53258
3	1112	0.35	24	275	55	243027	933	0.23	21	308	55	228903	3824	0.58	713	59	92079	3490	1.21	55	26	61	54303
4	986	0.49	27	290	64	291053	1011	0.32	22	308	67	233928	4087	0.51	827	74	99218	4339	0.88	56	12	68	55486
5	1052	0.82	27	303	77	292632	1233	0.41	24	310	77	258582	4388	0.77	893	78	104021	4389	1.02	59	13	79	57122

These results show that I^2CG_3 always finds the best solution in, generally, less computational time than the other algorithms for all tightness levels. The tightness of the side constraints has an impact on the optimality gap for most instances. Indeed, when many of those constraints are active (see their average percentage in column *Sc.*), they cut several interesting regions close to the best integer solutions and create many fractional extreme points. However, we can easily see that the effect of tight side constraints differs from one method to another. RMH is the mostly affected algorithm because it cannot find a feasible solution for some instances (two instances at level 4 and one instance at level 5). Notice also that the average computational times increase slightly with the tightness level, especially for the last three algorithms. As a conclusion, we can say that I^2CG_3 remains the best algorithm and that it is more stable against variations of the supplementary constraint right-hand side values.

5.7 Conclusion

In this paper, we introduced two new versions of the ICG algorithm, called I^2CG_2 and I^2CG_3 , which can solve the SPPSC. Because the side constraints in the SPPSC can break the quasi-integrality property of the SPP, the primal algorithms (e.g., ISUD), which pass only from one integer extreme point to an adjacent one, can get stuck in a local minimum. To overcome this limitation, we have added an artificial extreme point which is adjacent to all other extreme points of the feasible region convex hull in a higher dimensional space. Starting from this artificial extreme point, it becomes possible to find an integer direction leading to any integer extreme point of this polyhedron. The difference between algorithms I^2CG_2 and I^2CG_3 is that the latter takes into account the side constraints when searching for a descent direction while the former does not.

The proposed primal algorithms generate a sequence of integer solutions with decreasing costs until an optimal or near-optimal solution is reached. Computational experiments on CPP instances involving up to 1740 set partitioning constraints and 21 side constraints showed

that I^2CG_3 slightly outperforms I^2CG_2 and clearly perform better than two popular CG heuristics. For all tested instances, I^2CG_3 finds a large number of integer solutions and reaches a near-optimal solution in less computational time than the other algorithms, on average.

As an extension of this work, we intend to test the proposed algorithm on other types of problems with a variety of side constraints ($\leq$, $=$ and $\geq$) involving variable coefficients that may not be binary.

CHAPITRE 6 AMÉLIORATION DE LA MÉTHODE DE GÉNÉRATION DE COLONNES EN NOMBRES ENTIERS À L'AIDE DE LA PRÉDICTION DE SUCCESSIONS DE TÂCHES

6.1 Introduction

Dans ce chapitre, nous présentons la suite de nos travaux de recherche sur les méthodes primales de génération de colonnes en nombres entiers pour la résolution des problèmes de partitionnement d'ensembles avec ou sans contraintes supplémentaires. Les résultats présentés dans les chapitres précédents ont montré que la performance de ces méthodes dépend principalement de deux facteurs : l'efficacité de la méthode primale utilisée pour résoudre le RMP et la qualité de l'information primale (i.e., la succession de tâches) et duale (i.e., les coûts réduits) des colonnes générées.

Pour la résolution du RMP, la méthode I²CG décrite au chapitre précédent utilise une variante de l'algorithme ISUD. Cette dernière a montré son efficacité à trouver rapidement une suite décroissante de solutions entières menant à des solutions optimales ou quasi-optimales, malgré le nombre important de colonnes générées par les SPs. En effet, à chaque itération, I²CG cherche à générer des colonnes de bonne qualité i) duale, en utilisant une solution duale qui assure qu'au moins une variable non dégénérée dans une solution améliorée a un coût réduit négatif et ii) primale, en favorisant, à l'aide de la stratégie *multiphase*, les colonnes ayant une petite distance à la solution entière courante.

La stratégie *multiphase* se base sur l'information primale de la solution courante, pour chercher dans son voisinage des colonnes qui l'améliore. Malgré son importance en pratique, cette stratégie permet seulement d'explorer un espace restreint défini par un voisinage autour de la solution courante. Ceci signifie que I²CG trouverait une solution optimale seulement si les colonnes de cette solution appartiennent à ce voisinage. Autrement dit, pour que I²CG trouve rapidement la solution optimale ou quasi-optimale, il faut définir un voisinage auquel appartiennent les colonnes de la solution recherchée. Ce constat nous a donné l'idée de chercher les colonnes désirées dans le voisinage des meilleures solutions entières trouvées dans des résolutions antérieures. Dans le monde industriel, cet historique de solutions est souvent disponible en masse, mais généralement non exploité.

Dans ce chapitre, nous montrons que l'exploitation d'un historique de solutions entières du problème de rotations d'équipages peut être très utile pour l'amélioration de la performance des méthodes d'optimisation. En effet, nous proposons une nouvelle version de la méthode

I²CG dans laquelle nous générons de nouvelles colonnes en se basant sur des probabilités de connexions entre les vols. Ces probabilités sont calculées en utilisant un modèle de prédiction dont l'entraînement se fait sur l'historique des meilleures solutions entières disponibles.

La figure 6.1 montre le flux de données entre l'historique des solutions, le modèle de prédiction et la méthode de génération de colonnes en nombres entiers.

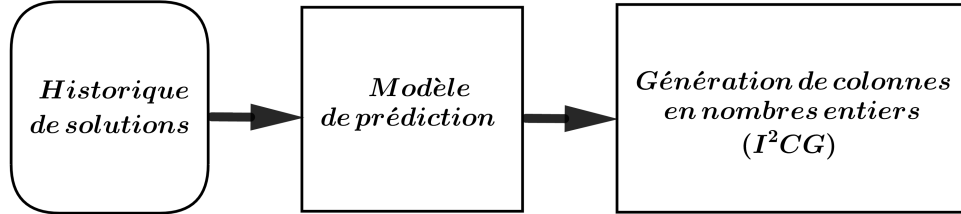


Figure 6.1 Diagramme de flux de données.

L'historique de solutions constitue une entrée du modèle de prédiction. Ce dernier produit les probabilités de connexion entre les vols, qui sont ensuite utilisées par I²CG au niveau des SPs pour générer les colonnes ayant une grande probabilité d'appartenir à une solution optimale ou quasi-optimale. En effet, lesdites probabilités permettent de restreindre l'espace de recherche au niveau des SPs en priorisant les connexions les plus prometteuses. Ceci permet à la fois de trouver rapidement des solutions entières de bonne qualité et de stabiliser les variables duales. Une idée se basant sur l'information duale a été proposée par Feillet et *al.* [43], l'essence en était de conduire la recherche de colonnes de coût réduits négatifs aux meilleures régions de l'espace en considérant dans le réseau du SP un sous-ensemble d'arcs avec coûts réduits attrayants.

Ce chapitre est organisé comme suit : nous présentons dans la section 6.2 une description de notre historique de solutions du CPP. Ensuite, nous décrivons dans la section 6.3 le modèle de prédiction que nous avons développé pour prédire les connexions entre les vols. Puis, nous présentons dans la section 6.4 la nouvelle version de la méthode I²CG dans laquelle nous utilisons les résultats du modèle de prédiction pour favoriser les colonnes qui ont plus de chance d'appartenir à une solution optimale ou quasi-optimale. Enfin, nous présentons dans la section 6.5 nos résultats numériques du modèle de prédiction et de la nouvelle version d'I²CG.

6.2 Historique de solutions

L'élaboration d'un modèle de prédiction quelconque nécessite un historique de données considérable sur lequel se fait l'entraînement de ce modèle. Dans un contexte d'application, la

compagnie aérienne possède des solutions obtenues lors des résolutions des mois passés. Malheureusement, nous n'avons pas accès à ces solutions. Donc, nous avons construit notre historique de solutions en résolvant 49 instances du problème de rotations d'équipage. Ces instances sont classées en 7 catégories selon le type d'avion. Dans chacune de ces instances, nous cherchons à affecter les vols planifiés aux équipages de la compagnie aérienne.

La résolution de ces instances nous donne une liste des rotations d'équipages permettant de couvrir tous les vols de la période, qui varie entre 4 et 7 jours, en minimisant les coûts. La figure 6.2 présente un exemple de rotation d'un équipage.

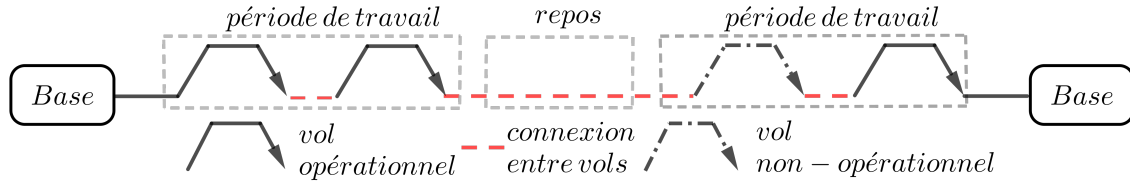


Figure 6.2 Exemple d'une rotation d'un équipage.

Dans cette rotation, l'équipage effectue une période de travail comprenant deux vols opérationnels (i.e., assurés par le même équipage), suivis d'une période de repos long. Enfin, l'équipage change d'aéroport à l'aide d'un vol non-opérationnel (i.e., l'équipage voyage comme passager), pour effectuer un vol opérationnel permettant de rentrer à la base.

L'analyse des meilleures solutions trouvées a montré que dans 90% des connexions, l'équipage opère un vol qui part de l'aéroport d'arrivée du vol précédent. En se basant sur ce constat, nous avons construit une liste de vols opérationnels $\mathcal{D}_i$ (ordonnée selon la date et l'heure de départ) qu'un équipage peut couvrir dans son horaire, juste après avoir assuré le vol i . La figure 6.3 montre une illustration d'un scénario de connexions possibles entre un vol entrant à un aéroport et des vols sortant du même aéroport.

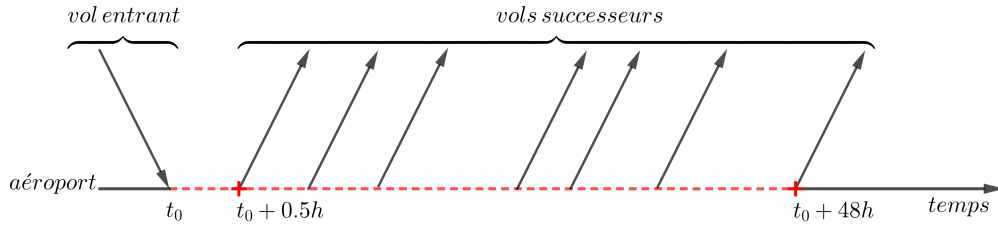


Figure 6.3 Illustration d'un scénario des vols successeurs d'un vol.

Il est à noter que dans notre contexte, le temps de connexion autorisé entre un vol entrant i et son successeur j est au moins de 30 minutes et ne dépasse pas 48 heures. Donc, dans la liste $\mathcal{D}_i$, on considère tous les vols j qui partent de l'aéroport d'arrivée du vol i et qui respectent

la condition sur le temps de connexion. Les connexions entre les vols opérationnels et non-opérationnels seront donc ignorées dans le modèle de prédiction. Par contre, ils seront traités dans la méthode de génération de colonnes en nombres entiers. Chaque vol opérationnel est identifié par les caractéristiques mentionnées dans le tableau 6.1.

Tableau 6.1 Les caractéristiques des vols.

Caractéristique	Type
Code du vol	catégorique
Type d’avion	catégorique
Aéroport de départ	catégorique
Date et heure du départ	date/heure
Aéroport d’arrivée	catégorique
Date et heure d’arrivée	date/heure

Il est à noter que dans nos instances, nous avons 7826 codes de vols, 7 types d’avions et 54 aéroports dont 3 sont des bases. Les codes ne sont pas uniques, c’est-à-dire que deux vols opérés la même journée par deux types d’avions différents peuvent avoir le même code. Les dates (de départ et d’arrivée) incluent aussi l’information sur l’heure et la minute (de départ et d’arrivée).

6.3 Modèle de prédiction

Dans cette section, nous décrivons le modèle de prédiction que nous avons construit pour prédire les connexions entre les vols. En considérant que chaque vol $j \in \mathcal{D}_i$ est associé à une classe k , où $k \in \{1, \dots, |\mathcal{D}_i|\}$ correspond au classement du vol j dans la liste $\mathcal{D}_i$, notre modèle de prédiction peut être perçu comme un modèle de classification multiclassés en apprentissage supervisé [25], où l’objectif est de prédire (avec une probabilité) la classe k du prochain vol $j \in \mathcal{D}_i$ que l’équipage d’une compagnie aérienne devrait suivre dans son emploi du temps après avoir effectué un vol entrant i . Puisque les réseaux de neurones ont montré une grande efficacité sur un problème semblable (Yaakoubi et *al.* [30]), nous avons opté pour un réseau de neurones comme modèle de prédiction. Il s’agit d’un réseau de neurones convolutionnel multicouches [25] où la couche de sortie est une couche dense avec *softmax* comme fonction d’activation. Pour chaque vol i , le réseau de neurones nous retourne donc un vecteur de probabilités associées aux vols successeurs $j \in \mathcal{D}_i$. Il est à noter que nous construisons 7 réseaux de neurones que nous entraînons pour produire les probabilités des connexions entre chaque paire de deux vols (i, j) , $i \in \mathbf{T}$ et $j \in \mathcal{D}_i$. Chaque réseau de neurones

est utilisé pour prédire les connexions entre les vols des instances d'un seul type d'avion. Pour ce faire, nous entraînons les réseaux de neurones d'une façon cyclique, c'est-à-dire que chaque réseau de neurones est entraîné en utilisant les données de 6 types d'avions pour produire les probabilités des connexions des instances du 7^{ième} type d'avion. Ce choix s'impose car nous ne disposons pas d'assez de données pour l'entraînement d'un seul modèle de prédiction efficace.

Pour concevoir un modèle de prédiction efficace, nous avons i) transformé et filtré les données décrites dans la section précédente afin de construire les données d'entrées du réseau de neurones ; ii) encodé les données à l'aide d'une couche cachée d'encodage ; iii) trouvé une architecture du réseau de neurones appropriée au CPP traité et iv) entraîné le réseau de neurones pour produire les probabilités de connexion entre les vols. Ces 4 étapes sont décrites respectivement dans les sous-sections qui suivent.

6.3.1 Transformation et filtrage des données

Les données décrites dans la section 6.2 sont utilisées pour construire les données d'entrée du modèle de prédiction. En effet, nous avons structuré les données sous forme de matrice, nommé $\mathbf{X}$, à trois dimensions $v \times n \times t$, où v est le nombre de vols entrants, n est le nombre maximum de vols successeurs qu'un vol entrant peut avoir et t est le nombre d'attributs utilisés pour concevoir le modèle de prédiction. Ces attributs sont déduits à partir des caractéristiques des vols présentées dans le tableau 6.1, en ignorant les codes des vols. Les dates (de départ et d'arrivée) sont transformées en deux attributs : l'heure et la minute (de départ et d'arrivée). À partir des dates, nous avons aussi déduit aussi la durée des vols et le temps de connexion entre les vols entrants et leurs successeurs. Un autre attribut "*Base*" a été ajouté pour indiquer la base à laquelle appartient l'équipage qui a couvert le vol entrant. À la matrice, $\mathbf{X}$, nous avons associé un vecteur $\mathbf{Y}$ de dimension v où chaque élément $\mathbf{y}_i$ indique la classe du vol $j' \in \mathcal{D}_i$, où j' est le successeur du vol i dans une solution de l'historique. Il est à noter que pour la prédiction du vol successeur, nous utilisons seulement l'information sur le vol entrant i , autrement dit, nous omettons les informations des vols antérieurs.

Nos premiers tests nous ont montré que la performance du modèle de prédiction est directement influencée négativement par le grand nombre de successeurs de chaque vol. En fait, les données dont on dispose sont insuffisantes pour entraîner un modèle efficace de classification multiclassées avec un grand nombre de classes possibles. Pour avoir une meilleure performance, nous avons réduit l'ensemble de successeurs $\mathcal{D}_i$ de chaque vol i , en éliminant les candidats $j \in \mathcal{D}_i$ improbables de suivre le vol i . Pour ce faire, nous réduisons la liste des aéroports destinations des vols successeurs, en se basant sur certains attributs des vols entrants. L'idée

ici consiste à éliminer le vol j , successeur d'un vol entrant i , dont l'aéroport de destination n'a jamais été visité par un équipage qui a effectué le vol i (dans les solutions de l'historique). Bien que des vols successeurs sont ignorés durant l'entraînement du modèle de prédiction, ils sont considérés au cours de la résolution du CPP. Les attributs utilisés dans le filtre sont catégorisés en deux types : géographiques et temporels.

Les critères géographiques utilisés sont :

- La base à laquelle appartient l'équipage qui a couvert le vol entrant.
- L'aéroport de départ et d'arrivée du vol entrant.

Les critères temporels utilisés sont :

- La durée du vol entrant.
- Le jour et l'heure de départ et d'arrivée du vol entrant.

Notre choix de ces critères pour construire un filtre est justifié, car : i) à l'aide de ces critères nous établissons approximativement une carte géographique et temporelle que les compagnies aériennes possèdent naturellement ; ii) on cherche seulement à limiter la liste des aéroports qu'un équipage peut atteindre après avoir effectué un vol entrant i . Autrement dit, on ne vise pas le vol successeur qui a été choisi dans la meilleure solution ; iii) on ne possède pas assez de données : une seule solution par instance pour entraîner un modèle de prédiction qui sera capable d'apprendre des "patterns" sur les aéroports de destination des vols successeurs. Le tableau 6.2 montre des statistiques sur le nombre de vols successeurs avant et après le filtre.

Tableau 6.2 Résultats du filtre

	Maximum du nombre de successeurs	Moyenne du nombre de successeurs
Avant filtre	149	45.38
Après filtre	22	4.81

La figure 6.4 suivante donne une illustration des résultats de filtrage des successeurs d'un vol entrant.

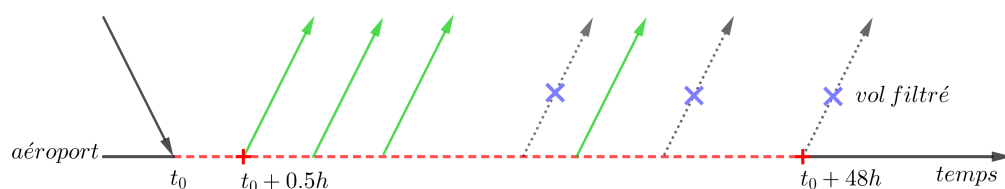


Figure 6.4 Illustration des vols après filtrage.

Dans la figure 6.4, les vols en vert représentent les vols qui ne sont pas enlevés par le filtre et les vols en pointillés sont les vols enlevés à l'aide du filtre. Dans la suite du document, nous

considérons pour chaque vol i , une liste $\mathcal{D}_i^f \subseteq \mathcal{D}_i$ contenant les vols non enlevés par le filtre qui sont ordonnés selon la date et l’heure de départ.

6.3.2 Couche cachée d’encodage

Avant de pouvoir utiliser les données décrites dans la sous-section précédente, un encodage des attributs catégoriques s’impose. Une première technique consiste à encoder ces attributs sous forme de valeurs numériques. Dans ce cas, même si les codes peuvent être attribués aléatoirement, les catégories du même attribut dont les valeurs sont proches seront traitées de la même façon par le modèle de prédiction. Afin d’éviter ce problème, nous avons encodé les attributs en utilisant une couche cachée d’encodage. Pour ce faire, nous avons attribué des valeurs numériques aux différentes catégories des attributs catégoriques. Ensuite, nous avons connecté chaque attribut aux neurones, 40 neurones au maximum, de la couche cachée d’encodage. Autrement dit, chaque catégorie d’un attribut peut être encodée à l’aide d’un vecteur de dimension $d \leq 40$ (d est un hyper-paramètre) et ce codage s’apprend au cours de l’entraînement du modèle. Par conséquent, toutes les valeurs d’un attribut donné sont encodées à l’aide d’une matrice de dimension $C \times d$ (où C est le nombre de valeurs catégoriques possibles).

Cette technique permet, en plus de la réduction de la dimension de représentation d’un attribut catégorique, de construire une entrée (des autres couches cachées) basée sur la similarité, où les vols qui ont des caractéristiques similaires auront un codage similaire. Cette similarité peut être efficacement exploitée par le réseau de neurones convolutionnel, d’où le choix de ce type des réseaux de neurones.

6.3.3 Architecture du réseau de neurones

Pour déterminer une architecture (configuration des hyper-paramètres) appropriée aux données de la compagnie aérienne traitée, nous avons défini un espace de recherche de configurations des hyper-paramètres permettant de construire un réseau de neurones ayant la meilleure performance possible. Pour ce faire, nous attribuons une plage de valeurs aux différents hyper-paramètres des réseaux de neurones. Ensuite, nous avons utilisé le processus Gaussien basé sur l’optimisation bayésienne (voir DERNONCOURT et *al.* [44]) avec la validation croisée en k -fois ($k=7$) sur les données d’entraînement pour chercher la meilleure configuration possible dans l’espace défini. À chaque itération, le processus Gaussien cherche à déterminer une architecture qui maximise la précision (accuracy en anglais) moyenne (pour les 7 types d’avions) du réseau de neurones. Les plages de valeurs des hyper-paramètres sont données dans le tableau 6.3.

Tableau 6.3 Les plages de valeurs des hyper-paramètres utilisés dans l’optimisation.

Hyper-paramètre	Plage de valeurs	Type
Algorithme d’optimisation	adadelta, adam, adagrad, RMSprop, SGD, Adamax, Nadam	Catégorique
Taux d’apprentissage	0.001, 0.002, $\dots$, 0.01	Réel
Dimension de l’encodage	5, 10, 15, $\dots$, 40	Entier
Nombre des couches denses	1, 2, 3, 4, 5	Entier
Nombre de neurones par couche	10, 60, $\dots$, 1000	Entier
Dropout rate	0.1, 0.2, $\dots$, 0.9	Réel
Nombre de couches convolutionnelles	0, 1, 2, 3	Entier
Nombre de filtres	50, 10, 250, $\dots$, 100	Entier
Taille des filtres des couches convolutionnelles	2, 3, 4, 5	Entier

Après l’identification de la meilleure configuration des hyper-paramètres, nous construisons 7 réseaux de neurones (ayant la même architecture) que nous entraînons pour produire les probabilités des connexions entre chaque paire de deux vols $i \in \mathbf{T}$ et $j \in \mathcal{D}_i^f$. Chaque réseau de neurones est utilisé pour produire les probabilités de connexion entre les vols des instances d’un seul type d’avion. Ces probabilités sont stockées dans une matrice de probabilités $\mathcal{P}$, où chaque élément $\mathcal{P}_{ij}$ représente la probabilité que le vol i soit suivi d’un vol $j \in \mathcal{D}_i$ dans un horaire d’un équipage. Il est à noter que nous imposons $\mathcal{P}_{ij} = 0$ pour tout $j \in \mathcal{D}_i \setminus \mathcal{D}_i^f$.

6.4 Génération de colonnes en nombres entiers améliorée

Dans cette section, nous décrivons la nouvelle version de notre algorithme de génération de colonnes en nombres entiers. Dans cette version, nous exploitons la matrice de probabilités $\mathcal{P}_{ij}$ pour favoriser la génération des colonnes (i.e., rotations des équipages) les plus probables d’appartenir à une solution optimale ou quasi-optimale. Pour ce faire, nous avons ajouté initialement une contrainte de ressource au niveau du SP, sa valeur est mise à jour à chaque connexion entre un vol entrant i et son successeur $j \in \mathcal{D}_i$, en multipliant $\mathcal{P}_{ij}$ par les probabilités de toutes les connexions précédentes. La dominance est aussi effectuée sur cette ressource pour favoriser les rotations ayant la plus grande valeur des produits de probabilités des connexions entre les vols. Nos premiers tests ont montré que le nouveau SP est plus difficile à résoudre à cause du grand nombre d’étiquettes non dominées.

Pour pallier ce problème, nous avons encodé les probabilités sous forme de nombres entiers en utilisant l’algorithme 4. L’entrée de ce dernier est : la matrice des probabilités $\mathcal{P}$ et la liste des successeurs possibles $\mathcal{D}_i$ de chaque vol entrant $i \in \mathbf{T}$. L’algorithme 4 nous retourne la matrice $\mathcal{O}$ et un entier *classMin*. Les éléments $\mathcal{O}_{ij}$ indiquent le classement du vol j dans la liste $\mathcal{D}_i$ ordonnée selon les probabilités $\mathcal{P}_{ij}$. L’entier *classMin* indique le classement

correspondant à la connexion la moins probable entre un vol entrant $i \in \mathbf{T}$ et un successeur $j \in \{k \in \mathcal{D}_i | \mathcal{P}_{ik} \neq 0\}$.

Algorithme 4: Encodage des probabilités

Entrées : matrice des probabilités $\mathcal{P}$, listes des successeurs $\mathcal{D}$
Sorties : matrice de classement $\mathcal{O}$, classement $classMin$

- 1 $classMin \leftarrow |\{j \in \mathcal{D}_i | \mathcal{P}_{ij} \neq 0\}|$, tel que $i \in \arg \max_{i \in \mathbf{T}} |\{j \in \mathcal{D}_i | \mathcal{P}_{ij} \neq 0\}|$;
- 2 $\mathcal{O}_{ij} \leftarrow +\infty$, pour tout $i \in \mathbf{T}$ et $j \in \mathbf{T} \setminus \mathcal{D}_i$;
- 3 $\mathcal{O}_{ij} \leftarrow classMin + 1$, pour tout $i \in \mathbf{T}$ et $j \in \mathcal{D}_i$;
- 4 **pour** $i = 1$ à $|\mathbf{T}|$ **faire**
- 5 Liste $\mathcal{V} \leftarrow$ trier les vols $j \in \mathcal{D}_i$ selon un ordre décroissant de $\mathcal{P}_{ij}$;
- 6 **pour** $k = 1$ à $|\mathcal{V}|$ **faire**
- 7 $j \leftarrow \mathcal{V}_k$;
- 8 **si** $\mathcal{P}_{ij} \neq 0$ **alors**
- 9 $\mathcal{O}_{ij} \leftarrow k$;

À l'itération 1, nous déterminons $classMin$ qui est égale au cardinal de l'ensemble $\{j \in \mathcal{D}_i | \mathcal{P}_{ij} \neq 0\}$ qui a le plus grand nombre de successeurs de probabilités non nulles. Ensuite, pour chaque vol entrant i , nous ordonnons les vols successeurs $j \in \mathcal{D}_i$ selon un ordre décroissant de probabilités $\mathcal{P}_{ij}$. Puis, nous attribuons, à chaque vol successeurs j , un classement $\mathcal{O}_{ij} = k$ tel que $\mathcal{P}_{ij} \neq 0$, $k \in \mathbb{N}$ et $k \leq classMin$. Enfin, nous attribuons le classement $classMin + 1$ à tous les vols successeurs $j \in \mathcal{D}_i$ dont la probabilité $\mathcal{P}_{ij} = 0$. Les deux sorties de l'algorithme 4 sont utilisées dans l'algorithme 5 au niveau du SP pour favoriser la génération des colonnes ayant une grande probabilité d'appartenir à une solution optimale ou quasi-optimale. La façon dont elles sont utilisées est décrite ci-bas.

L'algorithme 5 de la nouvelle version I²CG, nommé I²CG_p, se compose aussi de trois phases importantes : la génération de colonnes, la boucle RP-CP et Zoom. Il est à noter que l'algorithme 5 est conçu à partir de l'algorithme 3 en lui ajoutant deux étapes (31 et 34) et deux paramètres au SP (voir étape 3). Dans cette section, nous décrivons principalement la phase de génération de colonnes, les deux autres phases sont décrites en détail dans la section 5.5.

1. La génération de colonnes :

Cette phase consiste à générer un ensemble $\mathbf{J}$ de colonnes en résolvant, à l'aide de la programmation dynamique, un ou plusieurs SPs dont les coûts réduits des arcs sont calculés en utilisant la solution duale du CP3. Les nouvelles colonnes sont ajoutées à l'ensemble de colonnes $\tilde{\mathbf{N}}$. Des résultats théoriques relatifs à la solution duale π sont présentés dans le chapitre précédent.

Algorithme 5: I^2CG_p

Entrées : colonnes initiales $\tilde{\mathbf{N}}$, solution x^0 ($\mathcal{S}^0$), solution duale π^0 , coût z^0 , matrices des classements $\mathcal{O}$ et $classMin$

Sortie : meilleure solution entière trouvée x^*

Paramètres: CPversion, maxZoom, kZoom, minImprPerc, maxSuccFail, $\mathbf{K}^{CP} = \{k_1, \dots, k_\eta\}$

1 $r \leftarrow 0$; $nbSuccFail \leftarrow 0$; $x^* \leftarrow x^0$; $\mathcal{S} \leftarrow \mathcal{S}^0$; $classM \leftarrow classMin$;

2 **répéter**

Génération de Colonnes

3 $\mathbf{J} \leftarrow \text{résoudre SP}(\pi^r, \mathcal{O}, classM)$; $r \leftarrow r + 1$;

4 **si** $\mathbf{J} \neq \emptyset$ **alors**

5 $\tilde{\mathbf{N}} \leftarrow \tilde{\mathbf{N}} \cup \mathbf{J}$;

Boucle RP-CP

6 $k \leftarrow k_1$;

7 **répéter**

8 $z^r \leftarrow \text{résoudre RP2}(\mathcal{S})$;

9 **répéter**

10 $(d, \pi^r) \leftarrow \text{résoudre CPversion}(\tilde{\mathbf{N}}_{\mathcal{S}}^k)$;

11 **si** d est entière **ou** contient une sous direction entière **alors**

12 $\quad$ mettre à jour x^* et $\mathcal{S}$;

13 $\quad$ sortir;

14 $\quad k \leftarrow \text{prochainePhase}(k, \mathbf{K}^{CP})$;

15 **tant que** $k \neq 0$;

16 **tant que** d est entière **ou** contient une sous direction entière;

Zoom

17 **si** d est fractionnaire **alors**

18 $\mathcal{S}' \leftarrow \mathcal{S}$; $d^1 \leftarrow d$;

19 **pour** $\ell = 1$ à $maxZoom$ **faire**

20 $\mathcal{S}' \leftarrow \mathcal{S}' \cup \text{supp}(d^\ell)$;

21 $z^r \leftarrow \text{solve RP2}(\mathcal{S}')$;

22 **si** solution entière améliorée est trouvée **alors**

23 $\quad$ mettre à jour x^* et $\mathcal{S}$;

24 $\quad$ sortir;

25 **si** $\ell < maxZoom$ **alors**

26 $\quad d^{\ell+1} \leftarrow \text{solve CPversion}(\tilde{\mathbf{N}}_{\mathcal{S}'}^{k_{Zoom}})$;

27 **si** $d^{\ell+1} = d^\ell$ **alors**

28 $\quad$ sortir;

Contrôle

29 **si** $\frac{(z^{r-1} - z^r)}{z^{r-1}} < minImprPerc$ **alors**

30 $\quad nbSuccFail \leftarrow nbSuccFail + 1$;

31 $\quad classM \leftarrow classMin + 1$;

32 **sinon**

33 $\quad nbSuccFail \leftarrow 0$;

34 $\quad classM \leftarrow classMin$;

35 **tant que** $\mathbf{J} \neq \emptyset$ **et** $nbSuccFail < maxSuccFail$;

En plus de la solution duale π , nous passons également deux paramètres $\mathcal{O}$ et $classM$ au SP pour contrôler les connexions (entre les vols) à autoriser. En effet, dans la fonction d'extension, nous autorisons une connexion entre deux vols $i \in \mathbf{T}$ et $j \in \mathcal{D}_i$ si et seulement si $\mathcal{O}_{ij} \leq classM$. Autrement dit, une étiquette qui a couvert un vol i est étendue sur un arc qui couvre le vol j si et seulement si le classement $\mathcal{O}_{ij}$ est inférieur ou égal à $classM$. Étant donné que la valeur de $classM$ peut être soit $classMin$ ou bien $classMin + 1$ (voir les étapes 31 et 34), deux cas se présentent. Dans le premier cas, nous autorisons seulement les connexions probables entre les vols $i \in \mathbf{T}$ et $j \in \mathcal{D}_i$ (i.e., $\mathcal{P}_{ij} \neq 0$). De cette façon, nous réduisons le graphe du SP à un sous-graphe défini par un voisinage autour des meilleures solutions de l'historique. Tandis que dans le deuxième cas, nous autorisons toutes les connexions possibles entre les vols, incluant les connexions entre les vols $i \in \mathbf{T}$ et $j \in \mathcal{D}_i \setminus \mathcal{D}_i^f$. Le but est d'élargir l'espace de recherche des nouvelles colonnes permettant d'améliorer la solution courante.

Dans le SP, nous avons aussi ajouté une contrainte de ressource, dont la valeur est initialisée à 1, que nous mettons à jour lorsqu'une étiquette est étendue, en multipliant son ancienne valeur par $\mathcal{O}_{ij}$. La dominance est aussi effectuée sur cette ressource. Dans le cas où un vol i est suivi d'un vol non-opérationnel, on multiplie l'ancienne valeur de la ressource par $classM + 1$. Le but est de pénaliser les vols non-opérationnels qui coûtent plus cher aux compagnies aériennes. Il est à noter qu'aucune borne supérieure n'est imposée à cette contrainte de ressource.

2. **Boucle RP-CP** : Cette étape consiste à résoudre une suite des RP et CP pour trouver la meilleure solution entière possible. Comme dans l'algorithme 3, la version du CP à utiliser est passée en paramètre ($CPversion$). Pour le problème traité dans ce chapitre, nous utilisons CP3 qui a montré son efficacité à trouver des directions et sous-directions de descente entières. Une étude théorique permettant de caractériser ces directions de descentes est présentée dans le chapitre précédent. La boucle RP-CP est arrêtée lorsque le CP ne trouve pas de direction ou sous-direction entière.
3. **Zoom** : Lorsque la dernière direction de descente trouvée par le CP est fractionnaire, alors une recherche de solutions améliorées est effectuée dans le voisinage de cette direction. Pour ce faire, nous augmentons l'ensemble $\mathcal{S}$ de telle sorte que les colonnes correspondantes aux variables entrantes $v_j > 0$ soient compatibles avec la nouvelle solution désagrégée $\mathcal{S}'$ (étape 20). Ensuite, on résout le RP2 construit à l'aide des colonnes compatibles avec $\mathcal{S}'$. Si une solution entière est trouvée alors on met à jour x^* et l'ensemble $\mathcal{S}$. Sinon, on résout le problème complémentaire (étape 26) pour trouver une nouvelle direction fractionnaire permettant d'élargir le voisinage dans lequel nous cherchons une solution améliorée. L'élargissement du voisinage est autorisé $maxZoom$

fois. Donc, cette phase se termine si le *maxZoom* est atteint, une solution améliorée est trouvée, ou si le CP ne trouve pas de direction de descente.

Les trois phases précédentes sont répétées tant que : le SP génère des variables de coût réduit négatif et le nombre d'échecs successifs (*maxSuccFail*) d'amélioration de la solution x^* n'est pas dépassé. Une amélioration est considérée comme échec lorsque la nouvelle valeur objective z^r ne décroît pas d'un certain pourcentage (*imprPerc*) de l'ancienne valeur objective z^{r-1} (étape 29). Dans ce cas, nous incrémentons le nombre d'échecs (*nbSuccFail*) et nous réinitialisons *classM* à *classMin*+1. Lorsque l'amélioration est suffisante, nous réinitialisons *classM* à *classMin* et *nbSuccFail* à 0.

6.5 Expérimentation

Dans cette section, nous présentons les résultats numériques des modèles de prédiction et de la nouvelle version de l'algorithme de génération de colonnes en nombres entiers I^2CG_p . Dans les deux parties, nous avons utilisé des instances réelles du problème de rotations d'équipages, ayant jusqu'à 1740 vols. En plus des 7 instances utilisées dans le chapitre précédent (voir tableau 5.2), nous avons ajouté 42 nouvelles instances, donnant 35426 vols au total. Chaque instance concerne un type d'avion, trois bases et un horizon qui varie entre 4 et 7 jours. À ces instances, nous avons ajouté des contraintes supplémentaires pour limiter le nombre de pilotes par base et par jour. Nous rappelons que les 49 instances ont été réparties en 7 catégories d'instances selon le type d'avion. Pour chaque catégorie, nous avons entraîné le modèle de prédiction en utilisant les autres catégories. Il est à noter que la dernière ligne de chacun des tableaux présentés dans cette section représente la moyenne des résultats obtenus.

6.5.1 Résultats des modèles de prédiction

Dans cette sous-section, nous présentons les résultats relatifs aux modèles de prédiction. Comme nous l'avons mentionné précédemment, nous avons dans un premier temps cherché une architecture (configuration des hyper-paramètres) pour nos modèles de prédiction à l'aide du processus Gaussien. Pour implémenter ce dernier, nous avons utilisé la bibliothèque GPyOpt (version 1.2.1). À cette étape, nous avons trouvé 2500 architectures, parmi lesquelles nous avons choisi une architecture en se basant sur la précision moyenne que donne chaque architecture. Ensuite, nous avons implémenté l'architecture choisie et entraîné 7 modèles en utilisant les bibliothèques suivantes : Keras (Gulli et *al.* [45]) et Tensorflow (Abadi et *al.* [46]). Chaque modèle est utilisé pour la prédiction des connexions entre les vols des instances d'un type d'avion. Toutes nos expérimentations ont été réalisées sur une machine de 40 cœurs avec

384 Go de mémoire, permettant d'évaluer plusieurs modèles en parallèle.

Dans le tableau 6.4, nous présentons les résultats de performance des réseaux de neurones. Nous lisons de gauche à droite : le type d'avion, le nombre total des vols de chaque type d'avion, précisions pour *Top k* ($k = 1, 2, \dots, 7$) en % pour chaque type d'avion. Chaque colonne *Top k* montre la certitude en % que chaque probabilité $\mathcal{P}_{i_1 j_1}$, attribuée à une connexion réelle (effectuée dans les solutions de l'historique) entre les vols i_1 et $j_1 \in \mathcal{D}_{i_1}$, soit parmi les k plus grandes probabilités $\mathcal{P}_{i_1 j}$, $j \in \mathcal{D}_{i_1}$.

Tableau 6.4 Précision du réseau de neurones.

<i>Avion</i>		<i>Précision</i>						
<i>Type</i>	<i>Vols</i>	<i>Top 1</i>	<i>Top 2</i>	<i>Top 3</i>	<i>Top 4</i>	<i>Top 5</i>	<i>Top 6</i>	<i>Top 7</i>
727	1228	89.272	99.106	100.000	100.000	100.000	100.000	100.000
09	1826	92.979	99.819	100.000	100.000	100.000	100.000	100.000
94	2248	89.300	99.358	99.786	100.000	100.000	100.000	100.000
95	6796	78.273	96.105	99.385	99.931	100.000	100.000	100.000
757	6890	66.210	87.517	96.130	99.221	99.787	99.905	100.000
319	7070	79.701	96.130	99.805	100.000	100.000	100.000	100.000
320	9368	76.597	94.544	99.099	99.932	99.983	100.000	100.000
		81.761	96.082	99.172	99.869	99.967	99.986	100.000

Le tableau 6.4 montre que nos réseaux de neurones sont capables de prédire à 100% de certitude avec *Top 7* les connexions entre les vols entrants i et leurs successeurs $j \in \mathcal{D}_i$. En effet, nous avons obtenu : 100% de certitude avec *Top 3* pour les petites instances 727 et 09, 100% de certitude avec *Top 4* pour les instances 94 et 319, 100% de certitude avec *Top 5* pour l'instance 95, 100% de certitude avec *Top 6* pour l'instance 320 et 100% de certitude avec *Top 7* pour l'instance 757. Ces résultats montrent que les réseaux de neurones des petites instances sont plus précis. Ceci n'est pas surprenant car l'entraînement de ces réseaux de neurones se fait avec 97% des données de l'historique alors que l'entraînement du réseau de neurones d'un type d'avion ayant un grand nombre de vols (e.g., 320) se fait seulement avec 74% des vols des données.

6.5.2 Résultats d'I²CG_p

Dans cette sous-section, nous présentons les résultats des tests de notre méthode d'I²CG_p sur les instances de CPP. Ces résultats sont comparés à ceux de la version standard d'I²CG et la méthode DH (Diving Heuristic). DH est une méthode *Branch-and-Price* heuristique qui consiste à brancher en profondeur sur la variable fractionnaire ayant la plus grande valeur jusqu'à l'obtention d'une solution entière. Il est à noter que DH ne garantit pas de trouver

une solution entière, car elle peut se retrouver avec un RMP non réalisable à la fin de la branche explorée.

Pour I^2CG , nous rapportons les résultats obtenus en utilisant la stratégie *multi-phase* dans le RMP et SP. Pour I^2CG_p , nous rapportons les résultats obtenus en utilisant la stratégie *multi-phase* dans le RMP et les probabilités (encodées à l'aide de l'algorithme 4) dans le SP. La solution initiale des instances a été créée en ajoutant des colonnes artificielles pénalisées par un coût "grand M", où chaque colonne couvre une seule tâche. Les paramètres de l'algorithme I^2CG_p ont été initialisés à $minImprPerc = 0.0025$, $maxSuccFail = 9$, $kZoom = 5$, $maxZoom = 2$, $\mathbf{K}^{CP} = \{1, 2, 3, 4, 5, 6, 7\}$ et $\mathbf{K}^{PS} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. La valeur du paramètre *classMin* est donnée par l'algorithme 4 dépendamment de l'instance.

Pour nos tests, nous avons utilisé une machine linux de 3.4 Ghz, i7. Le SP a été résolu par la programmation dynamique en utilisant le code de la librairie Boost 1.54. Le solveur commercial utilisé pour résoudre les RMP est CPLEX (version 12.6).

Dans les tableaux 6.5 et 6.7, nous rapportons les résultats de nos expérimentations sur les 7 instances du chapitre précédent et les 42 nouvelles instances, respectivement. Nous lisons de gauche à droite : le temps total de calcul (*Temps*) en secondes, l'écart d'optimalité en % entre le coût de la meilleure solution trouvée et le coût optimal de la relaxation continue (*Gap*), le nombre d'itérations de génération de colonnes (*Itr.*), le nombre total de solutions entières trouvées (*IntS*), le nombre de colonnes générées (*Cols*). Pour DH, le nombre de solutions entières n'est pas indiqué, car il est toujours égal à 1. Il est à noter que les valeurs optimales de la relaxation continue ne sont calculées que pour rapporter les *gaps* d'intégralité ; ainsi, le temps nécessaire pour les calculer n'est pas inclus dans le temps total.

Dans les tableaux 6.6 et 6.8, nous rapportons le détail des résultats (d' I^2CG et d' I^2CG_p) présentés, respectivement, dans les tableaux 6.5 et 6.7. L'objectif est d'analyser les composantes (i.e., RP2, CP3 et SP) des deux méthodes. Nous lisons de gauche à droite : le temps consommé par chaque composante (*Temps*) en secondes, le nombre de solutions entières trouvées par RP2 (*IntS*), le nombre de sous-directions et de directions entières trouvées par CP3 (*IntD*), le nombre d'itérations de génération de colonnes (*Itr.*).

Les résultats présentés dans le tableau 6.5 montrent qu' I^2CG_p trouve plus rapidement des solutions entières ayant un *gap* d'intégralité inférieur ou égal à celui de solutions trouvées par I^2CG . En effet, I^2CG_p réduit le temps calcul de 56% en moyenne. Cette réduction varie entre 37% pour l'instance 757 et 65% pour les instances 319 et 320. Cependant, I^2CG_p ne trouve que 43% (en moyenne) des solutions entières trouvées par I^2CG . Ceci est attendu, car I^2CG_p se limite au voisinage défini autour des meilleures solutions de l'historique pour générer les solutions entières. Nous remarquons aussi qu'en moyenne, I^2CG_p fait moins d'itérations

Tableau 6.5 Résultats d'I²CG, I²CG_p et DH pour les instances de base.

<i>Instances</i>		<i>I²CG</i>					<i>I²CG_p</i>					<i>DH</i>			
<i>No.</i>	<i>Tâches</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Cols.</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Cols.</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>Cols.</i>
727	239	8	0.35	20	66	19848	4	0.24	13	17	18175	9	0.35	59	9474
09	348	13	0.16	19	56	28438	6	0.16	13	13	23334	5	0.36	26	10695
94	424	36	0.19	18	101	57028	19	0.17	11	26	57251	38	0.45	71	19615
95	1255	1669	0.17	23	489	419540	875	0.12	23	240	562639	8052	0.22	903	171415
757	1290	855	0.01	17	402	337999	538	0.02	22	271	322896	3479	0.03	756	105320
319	1293	1542	0.11	23	462	283315	537	0.11	18	168	304348	4514	0.24	1170	133403
320	1740	1874	0.05	17	565	369921	644	0.04	18	209	338110	8126	0.08	1317	151183
<i>Moyenne</i>		857	0.15	19	306	216584	374	0.12	16	134	232393	3460	0.24	614	85872

qu'I²CG tandis que les nombres des colonnes générées par les deux méthodes restent comparables. Nous remarquons aussi qu'I²CG_p est 10 fois plus rapide que DH avec de meilleurs gaps.

Tableau 6.6 Détail des résultats d'I²CG et d'I²CG_p pour les instances de base.

<i>Instances</i>		<i>I²CG</i>						<i>I²CG_p</i>					
<i>No.</i>	<i>Tâches</i>	<i>RP2</i>		<i>CP3</i>		<i>SP</i>		<i>RP2</i>		<i>CP3</i>		<i>SP</i>	
		<i>Temps</i>	<i>IntS</i>	<i>Temps</i>	<i>IntD</i>	<i>Temps</i>	<i>Itr.</i>	<i>Temps</i>	<i>IntS</i>	<i>Temps</i>	<i>IntD</i>	<i>Temps</i>	<i>Itr.</i>
727	239	0.8	25	3.8	41	2.2	20	0.2	7	1.5	10	1.7	13
09	348	0.8	9	7.2	47	3.2	19	0.3	5	2.7	8	2.2	13
94	424	5.7	28	18.6	75	7.4	18	1.3	9	9.8	17	5.6	11
95	1255	335.4	112	983.4	377	270.4	23	51.2	30	451.7	210	311.5	23
757	1290	8.4	46	691.2	356	129.2	17	4.3	31	393.8	240	119.6	22
319	1293	423.5	130	941.7	332	127.0	23	30.3	28	375.8	140	108.6	18
320	1740	291.5	203	1337.2	362	177.6	17	7.5	38	503.1	171	110.3	18
<i>Moyenne</i>		152.3	79	569.0	227	102.4	19	13.5	21	248.3	113	94.2	16

À partir du tableau 6.6, nous remarquons que le temps consommé par RP2 d'I²CG_p représente seulement 9% du temps consommé par RP2 d'I²CG. De plus, le RP2 d'I²CG_p consomme en moyenne 3.5% du temps de résolution, tandis que le RP2 d'I²CG en consomme 17.7%. Nous remarquons aussi que malgré le nombre comparable de colonnes générées par les deux méthodes, le temps consommé par CP3 d'I²CG_p représente seulement 43% du temps consommé par CP3 d'I²CG. Cette remarquable réduction des temps de RP2 et CP3 de la méthode I²CG_p, peut être expliqué par la convergence rapide (voir la figure 6.5) de la méthode I²CG_p vers une solution optimale ou quasi-optimale. Cette convergence implique une limitation du nombre de colonnes intéressantes (au niveau du RP2 et CP3) en termes de coût réduit et de degré d'incompatibilité, ce qui permet d'accélérer la résolution du RP2 et CP3. Enfin, nous constatons que par rapport à I²CG, I²CG_p peut réduire le temps du SP jusqu'à 37% (pour l'instance 320).

La figure 6.5 illustre l'amélioration de la valeur de l'objectif des deux méthodes I²CG et I²CG_p en fonction du temps. Cette figure nous montre clairement qu'I²CG_p converge plus rapidement vers une solution de la même qualité ou parfois de meilleure qualité. En effet,

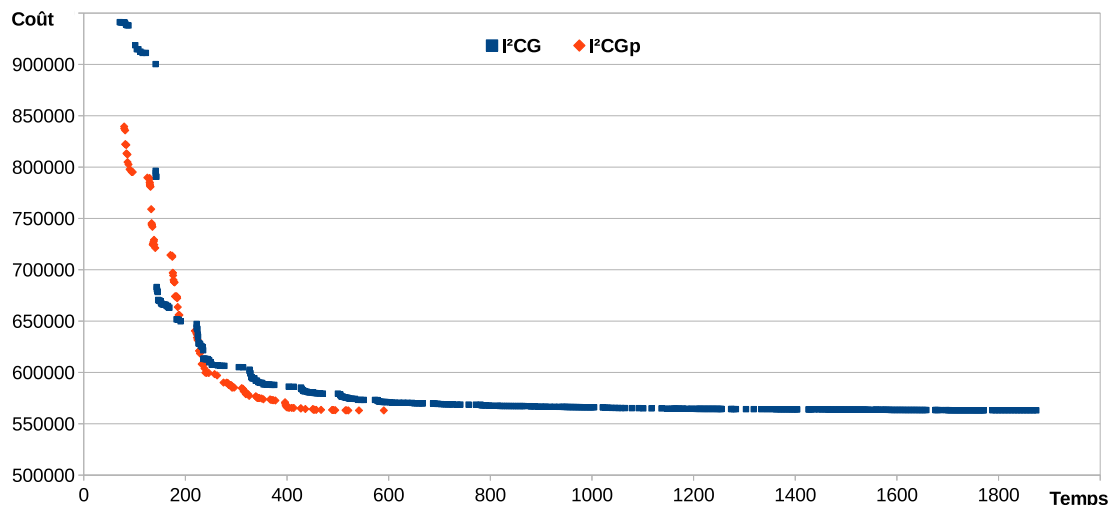


Figure 6.5 Le coût de la solution courante en fonction du temps pour l'instance 320.

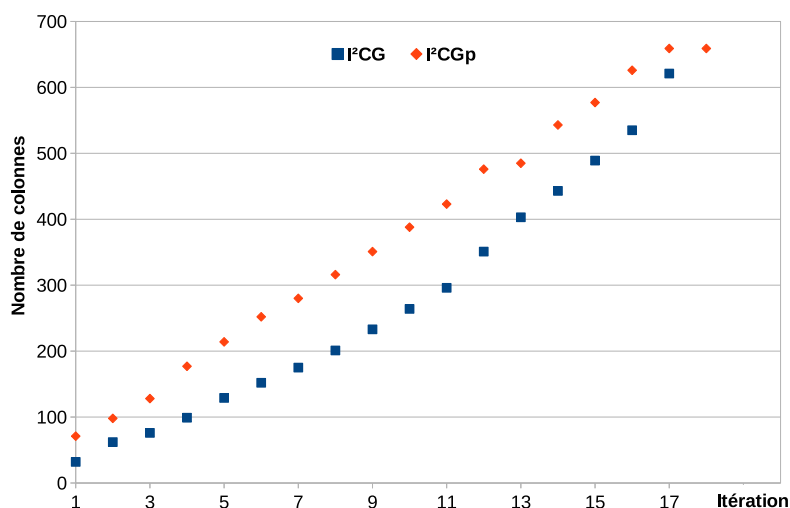


Figure 6.6 Le nombre de colonnes générées par itération de la solution de l'instance 320 .

au début de la résolution on constate que les deux méthodes ont un comportement similaire. Elles convergent vers la même solution dans les 200 premières secondes. Ensuite, I^2CG_p converge rapidement vers la meilleure solution, tandis qu' I^2CG converge plus lentement. La convergence d' I^2CG_p est expliquée par la génération rapide des solutions entières qui se trouvent dans le voisinage de la meilleure solution entière trouvée par I^2CG_p en fin de résolution. Cela permet de stabiliser les variables duales et de limiter le nombre de colonnes intéressantes en matière de coût réduit et de degré d'incompatibilité qu'il faut aller chercher. Parmi ces colonnes, on trouve celles de la meilleure solution recherchée. Par conséquent, I^2CG_p génère et trouve plus facilement des solutions entières quasi-optimales.

La figure 6.6 montre la progression de la génération des colonnes faisant partie de la meilleure

solution entière trouvée par les deux méthodes I^2CG et I^2CG_p . Nous constatons que durant toutes les itérations, le nombre de ces colonnes trouvées par I^2CG_p est plus grand que celui de la méthode I^2CG . Ceci n'est pas surprenant, car la méthode I^2CG_p génère les colonnes appartenant au voisinage défini autour des meilleures solutions de l'historique.

Tableau 6.7 Résultats d' I^2CG , I^2CG_p et DH pour les nouvelles instances.

<i>Instances</i>		<i>I²CG</i>					<i>I²CG_p</i>					<i>DH</i>			
<i>No.</i>	<i>Tâches</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Cols</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>IntS</i>	<i>Cols</i>	<i>Temps</i>	<i>Gap</i>	<i>Itr.</i>	<i>Cols</i>
727	234	9	0.07	22	78	18422	6	0.07	14	19	24149	5	0.13	28	8246
	169	5	0.02	23	53	12822	3	0.02	14	22	14468	3	0.15	23	5363
	167	6	0.00	24	61	14204	3	0.00	15	24	14805	2	0.00	18	5398
	154	5	0.17	21	46	10871	3	0.17	13	10	11323	2	0.56	23	4193
	154	9	0.14	23	52	11193	3	0.14	12	17	9033	3	0.55	33	4399
	111	2	0.00	20	43	5013	1	0.00	16	24	5120	1	0.00	22	2382
09	335	12	0.06	17	52	21725	8	0.06	14	8	24179	11	0.26	47	10502
	249	6	0.69	15	33	15747	7	0.20	15	17	19991	4	0.72	22	6655
	250	10	0.28	17	35	17713	6	0.28	12	12	18400	5	0.75	24	7171
	242	6	0.34	14	36	14248	6	0.17	14	13	17932	4	0.23	23	6690
	226	7	0.17	15	34	12617	4	0.05	12	6	15459	3	0.05	18	5631
	176	2	0.00	15	13	7129	1	0.00	11	16	7808	1	0.00	17	3682
94	415	30	0.28	20	87	51994	19	0.28	13	23	59654	65	0.64	129	19588
	304	15	0.23	15	62	30523	17	0.23	13	26	51282	33	0.60	95	12871
	304	26	0.36	17	71	37783	21	0.35	14	35	44964	42	1.23	111	13138
	292	18	0.02	20	72	33774	17	0.02	12	24	40157	13	0.03	36	11796
	287	14	0.09	15	71	33506	13	0.09	12	26	40118	11	0.15	31	11457
	222	9	0.25	21	84	23019	14	0.19	23	41	37734	10	0.24	57	7818
95	1216	1756	0.26	26	465	411655	662	0.21	21	156	544649	7834	0.31	774	159327
	907	933	0.14	24	425	326093	466	0.12	20	151	386780	5602	0.41	653	106911
	893	1380	0.21	26	430	319858	478	0.20	19	155	457220	5848	0.58	707	105148
	879	955	0.15	24	399	279692	452	0.13	21	124	367089	4079	0.27	497	92195
	868	1036	0.25	23	418	287513	427	0.18	21	141	373218	4827	0.52	554	103819
	778	758	0.11	23	443	187333	411	0.09	25	142	276225	4308	0.39	675	86868
757	1257	1183	0.03	19	327	308474	836	0.00	24	293	351226	5106	0.15	1077	115452
	918	593	0.00	20	309	233549	454	0.00	24	188	222362	1234	0.00	303	58284
	901	527	0.00	18	326	209662	486	0.00	22	233	256983	1645	0.00	408	64568
	900	596	0.00	18	332	221222	405	0.00	22	164	223593	2253	0.01	562	63853
	900	668	0.00	19	363	228250	497	0.00	20	195	249581	1625	0.00	359	62191
	724	228	0.00	18	175	130166	192	0.00	19	126	116261	391	0.00	175	34024
319	1263	1697	0.20	30	526	326464	495	0.18	18	147	391613	5424	1.06	1391	134446
	931	942	0.09	25	528	208332	360	0.06	22	157	221840	2442	0.31	817	78242
	933	1161	0.11	26	472	238510	338	0.11	20	165	272041	2344	0.22	682	78951
	939	803	0.09	23	522	223982	329	0.06	21	153	265147	2131	0.10	625	73695
	920	746	0.14	23	504	186920	298	0.11	19	149	245848	2979	0.67	1049	85855
	791	886	0.05	27	425	122349	258	0.05	20	263	157381	1206	0.34	763	59968
320	1694	1907	0.06	25	564	358081	794	0.09	21	253	405069	8488	0.20	1291	138385
	1236	869	0.04	22	467	254666	410	0.02	22	185	245035	3974	0.14	826	83903
	1257	942	0.03	21	491	257221	433	0.02	20	196	264950	4597	0.08	940	87237
	1216	761	0.02	21	414	227347	397	0.01	21	170	261814	2921	0.04	573	73090
	1204	801	0.05	22	436	216900	363	0.02	21	206	255706	3916	0.17	821	80735
	1021	462	0.01	20	291	153180	239	0.00	19	205	152642	325	0.00	113	36081
<i>Moyenne</i>		542	0.12	20	262	149755	253	0.09	17	111	176686	2040	0.29	414	52624

Les résultats des 42 nouvelles instances rapportés dans les tableaux 6.7 et 6.8 montrent qu' I^2CG_p trouve généralement de meilleures solutions en réduisant le temps d' I^2CG de 50%.

Tableau 6.8 Détail des résultats d'I²CG , I²CG et DH pour les nouvelles instances.

<i>Instances</i>		<i>I²CG</i>						<i>I²CG_p</i>					
<i>No.</i>	<i>Tâches</i>	<i>RP2</i>		<i>CP3</i>		<i>SP</i>		<i>RP2</i>		<i>CP3</i>		<i>SP</i>	
		<i>Temps</i>	<i>IntS</i>	<i>Temps</i>	<i>IntD</i>	<i>Temps</i>	<i>Itr.</i>	<i>Temps</i>	<i>IntS</i>	<i>Temps</i>	<i>IntD</i>	<i>Temps</i>	<i>Itr.</i>
727	234	0.7	18	4.3	60	2.8	22	0.3	7	3.0	12	2.2	14
	169	0.3	11	1.8	42	2.0	23	0.2	5	1.3	17	1.4	14
	167	0.8	13	2.3	48	2.2	24	0.1	5	1.2	19	1.5	15
	154	0.5	15	1.8	31	1.9	21	0.1	3	1.0	7	1.2	13
	154	1.9	14	3.5	38	1.8	23	0.5	3	1.4	14	1.0	12
	111	0.2	9	0.7	34	0.5	20	0.1	5	0.5	19	0.4	16
09	335	0.8	11	5.9	41	3.3	17	0.3	4	3.6	4	2.8	14
	249	0.3	7	2.9	26	2.3	15	0.3	6	3.2	11	2.3	15
	250	1.1	9	4.7	26	2.9	17	0.7	4	2.5	8	2.0	12
	242	0.3	8	2.6	28	2.1	14	0.2	5	2.7	8	2.2	14
	226	0.9	4	3.3	30	2.1	15	0.1	3	2.0	3	1.7	12
	176	0.2	4	0.5	9	0.8	15	0.1	4	0.5	12	0.6	11
94	415	2.3	24	13.6	63	10.3	20	1.0	8	9.0	15	7.3	13
	304	1.7	18	5.4	44	5.6	15	2.2	10	6.6	16	6.3	13
	304	6.3	19	10.0	52	6.8	17	3.7	7	8.4	28	6.5	14
	292	0.7	12	7.5	60	7.5	20	5.9	8	4.5	16	5.1	12
	287	0.5	12	6.1	59	5.8	15	1.1	14	4.9	12	5.2	12
	222	0.5	18	4.0	66	2.8	21	0.4	13	7.4	28	4.2	23
95	1216	328.5	87	956.0	378	394.1	26	29.7	21	291.1	135	299.1	21
	907	117.5	61	476.8	364	292.6	24	51.5	28	154.9	123	232.5	20
	893	249.6	55	686.0	375	395.7	26	51.5	31	163.2	124	233.9	19
	879	148.3	86	447.3	313	317.9	24	51.3	19	127.9	105	247.0	21
	868	216.5	75	480.8	343	292.2	23	9.2	26	157.2	115	236.4	21
	778	165.0	72	380.2	371	188.9	23	50.9	29	127.5	113	214.1	25
757	1257	74.9	74	831.0	253	246.6	19	5.3	39	511.9	254	286.8	24
	918	9.5	29	375.3	280	188.4	20	2.8	29	221.6	159	210.3	24
	901	7.2	32	337.0	294	166.5	18	3.9	29	247.1	204	216.2	22
	900	17.4	35	386.9	297	172.1	18	3.5	27	182.3	137	202.2	22
	900	7.7	35	457.9	328	183.2	19	3.7	28	252.5	167	220.2	20
	724	3.5	21	139.3	154	74.3	18	2.1	21	97.2	105	86.0	19
319	1263	149.4	107	1289.6	419	203.9	30	32.6	27	307.3	120	129.6	18
	931	34.1	39	742.5	489	138.0	25	38.4	24	177.7	133	125.3	22
	933	152.1	44	831.4	428	139.8	26	3.8	20	194.2	145	121.7	20
	939	64.8	72	581.8	450	126.7	23	3.2	17	195.2	136	112.2	21
	920	47.1	63	560.8	441	113.7	23	3.9	20	176.3	129	102.1	19
	791	127.7	93	673.5	332	67.8	27	2.0	29	196.2	234	49.4	20
320	1694	91.1	101	1503.3	463	255.5	25	7.0	35	545.5	218	212.4	21
	1236	12.4	39	663.6	428	160.2	22	9.3	26	241.8	159	143.5	22
	1257	32.3	64	699.4	427	179.9	21	4.0	22	259.8	174	151.2	20
	1216	6.1	32	566.3	382	164.7	21	4.5	24	239.0	146	135.9	21
	1204	17.9	38	614.9	398	142.2	22	3.6	30	220.4	176	122.5	21
	1021	5.7	33	374.4	258	67.2	20	2.2	21	160.6	184	66.3	19
<i>Moyenne</i>		50.1	38	360.4	224	112.7	20	9.4	17	131.2	93	100.2	17

En effet, par rapport à I²CG, I²CG_p réduit (en moyenne) le temps du RP2 de 81%, le temps de CP3 de 63% et le temps de SP de 11%. Nous constatons que pour atteindre la meilleure solution, I²CG_p a passé seulement par moins de 50% de points extrêmes entiers parcourus par I²CG. Néanmoins, on peut dire que par rapport à DH, I²CG_p trouve un grand nombre de solutions entières tout au long de la résolution, en moyenne 6 solutions entières par itération. De plus, I²CG_p améliore la solution entière courante à chaque 2 secondes

(en moyenne). Cette caractéristique des différentes versions d'ICG est très demandée dans l'industrie, car elle permet d'avoir une solution entière durant tout le processus de résolution. Par conséquent, la résolution peut être arrêtée à tout moment après avoir trouvé une première solution satisfaisante.

6.6 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche permettant d’explorer les techniques d’apprentissage machine pour améliorer la performance des méthodes d’optimisation. En effet, en utilisant l’historique de solutions, nous avons entraîné un réseau de neurones pour la prédiction des connexions entre les vols. Ce modèle nous a fourni les probabilités de connexions entre chaque paire de vols (i, j) , $j \in \mathcal{D}_i$.

En se basant sur les résultats du réseau de neurones, nous avons présenté une nouvelle version de l’algorithme I^2CG , nommée I^2CG_p . Dans cette dernière, nous avons utilisé les probabilités de connexion entre les vols pour favoriser la génération des colonnes ayant une plus grande probabilité d’appartenir à des solutions optimales ou quasi-optimales. Ces probabilités ont été encodées et utilisées uniquement au niveau de SPs pour définir un espace restreint de recherche de nouvelles colonnes. Cet espace correspond à un voisinage que nous avons défini autour des meilleures solutions de l’historique.

Les tests effectués sur 49 instances de CPP, impliquant jusqu’à 1740 vols, montrent clairement que la nouvelle méthode I^2CG_p est plus rapide que la version standard (I^2CG). I^2CG_p génère et trouve aussi une séquence de solutions entières avec des coûts décroissants jusqu’à atteindre une solution optimale ou quasi-optimale, en réduisant les temps de RP2, CP3 et SP.

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans le cadre de cette thèse, nous présentons trois travaux de recherche qui se complètent pour constituer un algorithme de génération de colonnes en nombres entiers permettant de résoudre des SPPs ou SPPSCs. Cet algorithme vient contribuer à l’essor des méthodes primales dont l’intérêt a été accentué avec les travaux cités auparavant. En effet, il apporte des améliorations aux deux composantes principales de la méthode CG, à savoir le RMP et le SP.

Au niveau du RMP, deux cas se présentent : i) si le RMP se modélise comme un SPP, notre algorithme le résout à l’aide d’ISUD, ii) sinon, le RMP est résolu à l’aide d’une variante de ISUD, lorsqu’il se modélise comme un SPPSC. Dans le premier cas, notre algorithme exploite les avantages de ISUD pour trouver facilement (i.e., souvent sans branchement) une séquence de solutions entières de coûts décroissants. En effet, ISUD profite de la dégénérescence du SPP et de sa quasi-intégralité. Malheureusement, les contraintes supplémentaires peuvent briser cette propriété intéressante qui est la base de plusieurs méthodes primales (i.e., IS et ISUD). Ce constat, représentant de véritables défis de recherche, était derrière l’amélioration de notre algorithme, qui se concrétise par la gestion des contraintes supplémentaires. Les deux variantes nous ont permis de résoudre efficacement le RMP et de trouver une suite de solutions entières menant à des solutions optimales ou quasi-optimales.

Au niveau du SP, notre contribution réside dans l’utilisation d’une solution duale dite correspondante à une solution entière courante pour définir le SP. En conséquence, ceci permet de générer des colonnes qui favorisent l’intégralité des directions de descente. Ce type de solution duale assure qu’au moins une variable non-dégénérée dans la solution améliorée a un coût réduit négatif. En utilisant cette solution duale, nous avons implémenté la stratégie multiphase au niveau du SP pour favoriser la génération des colonnes qui ont une petite distance par rapport à la solution courante. Cette stratégie permet de chercher de nouvelles colonnes dans le voisinage de la solution courante. Pour l’améliorer, nous avons proposé une nouvelle stratégie qui permet aussi de réduire l’espace de recherche de nouvelles colonnes à un sous-espace défini autour des meilleures solutions entières trouvées dans des résolutions antérieures. Cet historique de solutions a été exploré en utilisant des techniques d’apprentissage profond pour calculer les probabilités de connexions entre les vols dans le cas du CPP, pour ensuite générer les colonnes qui ont la plus grande probabilité d’appartenir à une solution optimale ou quasi-optimale.

Cette thèse constitue le premier travail scientifique qui illustre l’intérêt indéniable de l’utili-

sation des méthodes primales dans le contexte de génération de colonnes pour la résolution des SPPs et SPPSCs. Ceci se traduit non seulement par la qualité de la solution entière trouvée, mais également par la quantité des solutions entières rencontrées durant la résolution. Nos résultats numériques montrent que ces méthodes sont efficaces et permettent de générer beaucoup de solutions entières jusqu'à l'obtention d'une solution entière satisfaisante. En effet, cette caractéristique remarquable prouve que ce type de méthodes est plus adapté aux problèmes industriels difficiles où les compagnies cherchent parfois une seule solution réalisable. Nos résultats théoriques montrent que le choix de la solution duale est aussi primordial pour la génération des colonnes favorisant l'intégralité des directions de descente trouvées par le CP. D'autres résultats théoriques ont aussi été présentés pour caractériser des directions de descente. Nous tenons à noter que les contributions présentées dans cette thèse sont transférables et pourraient être adaptées et utilisées pour la résolution d'autres problèmes industriels.

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS

L'objectif général de cette thèse était de développer une nouvelle méthode de génération de colonnes en nombres entiers pour la résolution des problèmes dont le MP se formule comme un SPP ou SPPSC. Pour atteindre cet objectif, nous avons proposé progressivement trois variantes de la méthode primale ICG. Chaque variante a fait l'objet d'un sujet de recherche.

8.1 Synthèse des travaux

Dans le premier sujet, nous avons proposé un nouvel algorithme primal de génération de colonnes en nombres entiers, baptisée ICG. Ce dernier combine la génération de colonnes et l'algorithme ISUD pour résoudre les SPPs impliquant un très grand nombre de variables. Ce choix est motivé par les résultats numériques de cet algorithme qui trouve une suite de solutions entières menant à une solution optimale ou quasi-optimale en profitant de la dégénérescence du SPP. ICG bénéficie de la génération d'un grand nombre de colonnes à chaque itération de génération de colonnes et exploite une solution duale qui correspond à la solution entière courante pour favoriser la génération de colonnes des solutions entières améliorées. Pour contourner la première limitation de l'algorithme ISUD, ICG résout des petits SPPs pour chercher des solutions entières améliorées dans le voisinage de la solution entière courante. Cette technique ne contredit pas le paradigme primal et permet d'identifier d'autres solutions entières que ISUD ne trouve pas.

Les résultats prometteurs de l'algorithme ICG ont motivé l'introduction du deuxième algorithme I²CG pour résoudre le SPPSC. Comme nous l'avons mentionné précédemment, les contraintes supplémentaires peuvent briser la quasi-intégralité du SPP. En conséquence, les algorithmes primaux (e.g., ISUD), qui ne passent que d'un point extrême entier à un point adjacent, peuvent se coincer dans un minimum local. Pour surmonter ce problème, nous avons ajouté un point extrême artificiel qui est adjacent à tous les autres points extrêmes. À partir de ce point extrême artificiel, il devient possible de trouver une direction de descente entière menant à n'importe quel point extrême entier. Ces directions sont identifiées en utilisant un nouveau CP dans lequel nous considérons les contraintes supplémentaires. Une procédure de recherche des solutions entières dans le voisinage des directions fractionnaires a été aussi utilisée dans l'algorithme I²CG. Les résultats numériques des tests sur des instances de CPP montrent que I²CG trouve aussi des solutions optimales ou quasi-optimales en passant par plusieurs solutions entières intermédiaires.

Dans le troisième sujet, nous avons proposé une amélioration de l'algorithme I^2CG à l'aide des techniques d'apprentissage profond. Pour ce faire, nous avons entraîné un réseau de neurones artificiels sur un historique de solutions entières. Ce réseau de neurones prédit avec une certaine probabilité les connexions entre les vols. Ces probabilités ont été encodées et utilisées au niveau du SP pour définir un espace restreint de recherche de nouvelles colonnes. À partir de cet espace, I^2CG génère des colonnes ayant une plus grande probabilité d'appartenir à des solutions optimales ou quasi-optimales. Les résultats numériques montrent une réduction significative du temps de résolution du RP2, CP3 et SP.

8.2 Limitations des algorithmes proposés

Malgré les nombreux avantages des algorithmes proposés dans cette thèse, ils présentent certaines limitations à traiter dans le futur. Une première limitation concerne le calcul du degré d'incompatibilité dans la stratégie multiphase. Cette dernière consiste à sélectionner, parmi l'ensemble des colonnes générées, un sous-ensemble de colonnes à considérer dans le RMP selon leurs degrés d'incompatibilité. Ce dernier est calculé uniquement en se basant sur la succession des tâches couvertes par chaque colonne (i.e., les contraintes supplémentaires sont ignorées). Ceci signifie que la valeur du degré d'incompatibilité d'une colonne peut être biaisée lorsque le nombre des contraintes supplémentaires est élevé. Par conséquent, nos algorithmes risquent de ne pas choisir les bonnes colonnes, i.e., celles qui favorisent l'intégralité des directions de descente.

Une autre limitation concerne les formulations du SPP et SPPSC dans lesquels nous ne considérons pas des variables entières (i.e., non binaires). En effet, certains problèmes industriels utilisent ce type de variables pour modéliser certains aspects du problème traité. Dans le cas du CPP, des variables entières peuvent être utilisées pour calculer la sur-couverture et la sous-couverture des vols. Dans le cas des problèmes de tournées de véhicules, des variables entières sont utilisées pour minimiser le nombre de véhicules utilisés.

8.3 Améliorations futures

L'algorithme I^2CG a été testé uniquement sur des instances de CPP avec des contraintes supplémentaires $\leq$ et des coefficients binaires. Il serait donc intéressant d'effectuer, dans un premier temps, des tests sur d'autres types de problèmes avec une variété de contraintes supplémentaires ($\leq$, $=$ et $\geq$). Dans un second temps, I^2CG pourrait être généralisée pour le problème de recouvrement d'ensemble (où les contraintes de partitionnement sont remplacées par des contraintes $\geq$).

Dans des travaux de recherche futurs, plusieurs pistes de recherches peuvent être explorées pour améliorer nos algorithmes. Une première piste consiste à explorer les techniques d'apprentissage profond au niveau du RMP pour choisir les colonnes prometteuses à considérer dans le RP et CP. Ces techniques peuvent être aussi utilisées pour estimer les poids, de la contrainte de normalisation, qui pénalisent les directions de descente fractionnaires.

Une autre piste de recherche serait d'adapter notre algorithme pour qu'il soit capable de résoudre un programme linéaire en nombres entiers quelconque. Pour ce faire, nous pourrions généraliser dans un premier temps le CP3 introduit dans le chapitre 5.4. Ensuite, nous généraliserons la procédure de recherche des solutions entières dans le voisinage des directions de descente fractionnaires.

RÉFÉRENCES

- [1] E. Balas et M. Padberg, “On the set-covering problem : Ii. an algorithm for set partitioning,” *Operations Research*, vol. 23, n^o. 1, p. 74–90, 1975.
- [2] G. L. Thompson, “An integral simplex algorithm for solving combinatorial optimization problems,” *Computational Optimization and Applications*, vol. 22, n^o. 3, p. 351–367, 2002.
- [3] A. Saxena, “Set-partitioning via integral simplex method,” *Unpublished manuscript, OR Group, Carnegie-Mellon University, Pittsburgh*, 2003.
- [4] E. Rönnberg et T. Larsson, “Column generation in the integral simplex method,” *European Journal of Operational Research*, vol. 192, n^o. 1, p. 333–342, 2009.
- [5] —, “All-integer column generation for set partitioning : Basic principles and extensions,” *European Journal of Operational Research*, vol. 233, n^o. 3, p. 529–538, 2014.
- [6] A. Zaghroui, I. El Hallaoui et F. Soumis, “Integral simplex using decomposition for the set partitioning problem,” *Operations Research*, vol. 62, n^o. 2, p. 435–449, 2014.
- [7] —, “Improving set partitioning problem solutions by zooming around an improving direction,” *Annals of Operations Research*, p. 1–27, 2018.
- [8] —, “Improved integral simplex using decomposition for the set partitioning problem,” *EURO Journal on Computational Optimization*, vol. 6, n^o. 2, p. 185–206, 2018.
- [9] D. R. Morrison, S. H. Jacobson, J. J. Sauppe et E. C. Sewell, “Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning,” *Discrete Optimization*, vol. 19, p. 79–102, 2016.
- [10] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh et P. H. Vance, “Branch-and-price : Column generation for solving huge integer programs,” *Operations Research*, vol. 46, n^o. 3, p. 316–329, 1998.
- [11] G. B. Dantzig et P. Wolfe, “Decomposition principle for linear programs,” *Operations Research*, vol. 8, n^o. 1, p. 101–111, 1960.
- [12] A. N. Letchford et A. Lodi, “Primal cutting plane algorithms revisited,” *Mathematical Methods of Operations Research*, vol. 56, n^o. 1, p. 67–81, 2002.
- [13] P. C. Gilmore et R. E. Gomory, “A linear programming approach to the cutting-stock problem,” *Operations Research*, vol. 9, n^o. 6, p. 849–859, 1961.
- [14] J. Desrosiers, F. Soumis et M. Desrochers, “Routing with time windows by column generation,” *Networks*, vol. 14, n^o. 4, p. 545–565, 1984.

- [15] M. E. Lübbecke et J. Desrosiers, “Selected topics in column generation,” *Operations Research*, vol. 53, n°. 6, p. 1007–1023, 2005.
- [16] F. Vanderbeck, “Implementing mixed integer column generation,” dans *Column generation*. Springer, 2005, p. 331–358.
- [17] I. Elhallaoui, D. Villeneuve, F. Soumis et G. Desaulniers, “Dynamic aggregation of set-partitioning constraints in column generation,” *Operations Research*, vol. 53, n°. 4, p. 632–645, 2005.
- [18] I. Elhallaoui, A. Metrane, F. Soumis et G. Desaulniers, “Multi-phase dynamic constraint aggregation for set partitioning type problems,” *Mathematical Programming*, vol. 123, n°. 2, p. 345–370, 2010.
- [19] I. Elhallaoui, A. Metrane, G. Desaulniers et F. Soumis, “An improved primal simplex algorithm for degenerate linear programs,” *INFORMS Journal on Computing*, vol. 23, n°. 4, p. 569–577, 2011.
- [20] H. Bouarab, I. El Hallaoui, A. Metrane et F. Soumis, “Dynamic constraint and variable aggregation in column generation,” *European Journal of Operational Research*, vol. 262, n°. 3, p. 835–850, 2017.
- [21] V. Trubin, “On a method of solution of integer linear programming problems of a special kind,” dans *Soviet Mathematics Doklady*, vol. 10, 1969, p. 1544–1546.
- [22] E. Balas et M. W. Padberg, “On the set-covering problem,” *Operations Research*, vol. 20, n°. 6, p. 1152–1161, 1972.
- [23] S. Rosat, I. Elhallaoui, F. Soumis et A. Lodi, “Integral simplex using decomposition with primal cutting planes,” *Mathematical Programming*, vol. 166, n°. 1-2, p. 327–367, 2017.
- [24] S. Rosat, I. Elhallaoui, F. Soumis et D. Chakour, “Influence of the normalization constraint on the integral simplex using decomposition,” *Discrete Applied Mathematics*, vol. 217, p. 53–70, 2017.
- [25] I. Goodfellow, Y. Bengio et A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [26] W. Di, Y.-p. NIE et J.-m. HUANG, “Parametric dropout in rnn,” *DEStech Transactions on Computer Science and Engineering*, n°. smce, 2017.
- [27] A. M. Alvarez, Q. Louveaux et L. Wehenkel, “A supervised machine learning approach to variable branching in branch-and-bound,” dans *IN ECML*, 2014.
- [28] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser et B. Dilkina, “Learning to branch in mixed integer programming,” dans *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

- [29] R. Václavík, A. Novák, P. Šůcha et Z. Hanzálek, “Accelerating the branch-and-price algorithm using machine learning,” *European Journal of Operational Research*, vol. 271, n°. 3, p. 1055–1069, 2018.
- [30] Y. Yaakoubi, F. Soumis et S. Lacoste-Julien, “Flight-connection prediction for airline crew scheduling to construct initial clusters for or optimizer,” GERAD, Montréal, QC, Rapport technique G–2019(26), 2019.
- [31] Y. Bengio, A. Lodi et A. Prouvost, “Machine learning for combinatorial optimization : a methodological tour d’horizon,” *arXiv preprint arXiv :1811.06128*, 2018.
- [32] A. Lodi et G. Zarpellon, “On learning and branching : a survey,” *Top*, vol. 25, n°. 2, p. 207–236, 2017.
- [33] K. Haase, G. Desaulniers et J. Desrosiers, “Simultaneous vehicle and crew scheduling in urban mass transit systems,” *Transportation Science*, vol. 35, n°. 3, p. 286–303, 2001.
- [34] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. M. Solomon et F. Soumis, “Crew pairing at Air France,” *European Journal of Operational Research*, vol. 97, n°. 2, p. 245–259, 1997.
- [35] M. Saddoune, G. Desaulniers et F. Soumis, “Aircraft pairings with possible repetitions of the same flight number,” *Computers & Operations Research*, vol. 40, n°. 3, p. 805–814, 2013.
- [36] S. Irnich et G. Desaulniers, “Shortest path problems with resource constraints,” dans *Column generation*. Springer, 2005, p. 33–65.
- [37] C. Joncour, S. Michel, R. Sadykov, D. Sverdlov et F. Vanderbeck, “Column generation based primal heuristics,” *Electronic Notes in Discrete Mathematics*, vol. 36, p. 695–702, 2010.
- [38] O. J. Ibarra-Rojas, F. Delgado, R. Giesen et J. C. Muñoz, “Planning, operation, and control of bus transport systems : A literature review,” *Transportation Research Part B : Methodological*, vol. 77, p. 38–75, 2015.
- [39] A. Kasirzadeh, M. Saddoune et F. Soumis, “Airline crew scheduling : models, algorithms, and data sets,” *EURO Journal on Transportation and Logistics*, vol. 6, n°. 2, p. 111–137, 2017.
- [40] A. Tahir, G. Desaulniers et I. El Hallaoui, “Integral column generation for the set partitioning problem,” *EURO Journal on Transportation and Logistics*, p. 1–32, 2019.
- [41] R. E. Gomory, “All-integer integer programming algorithm,” *Industrial Scheduling*, 1963.
- [42] G. Desaulniers, J. Desrosiers et M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006, vol. 5.

- [43] D. Feillet, M. Gendreau et L.-M. Rousseau, “New refinements for the solution of vehicle routing problems with branch and price,” *INFOR : Information Systems and Operational Research*, vol. 45, n°. 4, p. 239–256, 2007.
- [44] F. Deroncourt et J. Y. Lee, “Optimizing neural network hyperparameters with gaussian processes for dialog act classification,” dans *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, p. 406–413.
- [45] A. Gulli et S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.
- [46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean et M. Devin, “Tensorflow : Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv :1603.04467*, 2016.

ANNEXE A

Cette annexe est associée à l'article présenté dans le chapitre 4.

A.1 Compatibility matrices

Variable substitution can be used to reduce the density of the constraint coefficient matrix of the CP (4.7)–(4.11). A first reformulation, proposed by Zaghrouti *et al.* [6], is obtained by isolating each λ_l variable in the last constraint (4.8) in which it appears before substituting it in the rest of the formulation. This substitution yields the following reformulation of the CP :

$$\min_v \quad (\mathbf{c}_{I_S}^\top - \mathbf{c}_S^\top (\mathbf{A}_S^1)^{-1} \mathbf{A}_{I_S}^1)^\top \mathbf{v} \quad (\text{A.1})$$

$$\text{s.t. :} \quad (\mathbf{A}_S^2 (\mathbf{A}_S^1)^{-1} \mathbf{A}_{I_S}^1 - \mathbf{A}_{I_S}^2) \mathbf{v} = 0 \quad (\text{A.2})$$

$$\sum_{j \in I_S} w_j v_j = 1 \quad (\text{A.3})$$

$$v_j \geq 0, \quad \forall j \in I_S \quad (\text{A.4})$$

$$v_{j_1} v_{j_2} = 0, \quad \forall (j_1, j_2) \in F_S, \quad (\text{A.5})$$

where, for $U \subset N$, $\mathbf{c}_U$ is the subvector of the cost coefficient vector in (4.7) associated with the variables v_j , $j \in U$. Constraint (A.2) can be written in the form $(\mathbf{M}_1 \mathbf{A}_{I_S})^\top \mathbf{v} = 0$, where $\mathbf{M}_1$ is a compatibility matrix according to the following definition.

Definition 11. A matrix $\mathbf{M}$ is said to be a compatibility matrix if and only if $\mathbf{M} \mathbf{A}_j = 0$ for all compatible columns $\mathbf{A}_j$, $j \in C_S$, and $\mathbf{M} \mathbf{A}_j \neq 0$ for all incompatible columns $\mathbf{A}_j$, $j \in I_S$.

There exists an infinite number of compatibility matrices that can be used to define the CP. The choice of this matrix may have a significant impact on the computational times. We have chosen to use the matrix $\mathbf{M}_2$ introduced by Bouarab *et al.* [20] that is specialized for vehicle routing and crew scheduling problems. In these problems, each column is associated with a route or a crew schedule that performs a subset of tasks in a given sequence. The $\mathbf{M}_2$ matrix allows to measure the deviation of the incompatible columns with respect to the task sequences defined by the columns in solution S . This matrix is such that, if an incompatible column $\mathbf{A}_j$, $j \in I_S$, does not respect the task ordering in a sequence, $\mathbf{M}_2 \mathbf{A}_j$ contains a component equal to -1 each time that $\mathbf{A}_j$ covers a task but not its predecessor and

a component equal to 1 each time that $\mathbf{A}_j$ covers a task but not its successor (see Bouarab *et al.* [20] for details).

A.2 Multi-phase strategy

Inspired from that developed by Elhallaoui *et al.* [18] in the context of DCA, Zaghrouti *et al.* [6] have elaborated a *multi-phase* acceleration strategy for ISUD which also aims at reducing the density of the CP constraint coefficient matrix. Each time that the CP needs to be solved, a sequence of phases can be invoked. Each phase is defined by a parameter k (a positive integer) which restricts the CP to a subset $I_S^k \subseteq I_S$ of the incompatible columns such that $I_S^k \subseteq I_S^\ell$ if $k < \ell$. The sequence of phases is predetermined and corresponds to an increasing sequence $k_1, k_2, \dots, k_p$ of values of k with $k_p = \infty$ (for instance, $k = 2, 3, 4, 5, 6, \infty$), where phase $k_p = \infty$ means that $I_S^\infty = I_S$. Starting in phase k_1 , the CP restricted to the subset $I_S^{k_1}$ is solved. If its optimal value is negative, the process stops and the computed linear combination of incompatible columns is returned. Otherwise, the next phase is invoked. The process repeats until obtaining a negative optimal value for the CP or reaching phase $k_p = \infty$.

The definition of a subset I_S^k used in phase k is based on a distance between an incompatible column and the vector subspace generated by the columns in the solution. This distance, called the degree of incompatibility, is defined as follows.

Definition 12. *The degree of incompatibility δ_j of an incompatible column $\mathbf{A}_j$, $j \in I_S$, is given by $\delta_j = \|\mathbf{M}\mathbf{A}_j\|$, where $\mathbf{M}$ is a compatibility matrix (as defined in Appendix A.1).*

In phase k of the multi-phase strategy for solving the CP, the index subset of the incompatible columns considered in the CP is defined by $I_S^k = \{j \in I_S \mid \delta_j \leq k\}$. Bouarab *et al.* [20] has proven the following result.

Proposition 7. *In phase k , each incompatible column $\mathbf{A}_j$, $j \in I_S^k$, has at most $k+1$ non-zero coefficients in the constraint coefficient matrix of the CP if $\mathbf{M}_2$ is used as the compatibility matrix.*

Consequently, for phases with a low k value, the constraint coefficient matrix has a low density. This helps to reduce the computational effort for solving the CP and also to produce integer directions without branching. Finally, it may allow to consider a very large number of incompatible columns in the CP if many are available.

A.3 Computation of a complete dual solution

To compute a complete dual solution $\boldsymbol{\alpha} = (\alpha_i)_{i \in T} \in \mathbb{R}^m$ to the MP, we use as in Bouarab *et al.* [20] the dual variable values $\hat{\boldsymbol{\pi}} \in \mathbb{R}^{m-|S|}$ associated with constraints (A.2) that were computed when solving the last relaxed CP (A.1)–(A.4) (i.e., without constraints (A.5)). Recall that, for each column index $j \in S$, there are $|T_j| - 1$ constraints (A.2) in the CP, where T_j is the subset of tasks covered by $\mathbf{A}_j$. These constraints are associated with the first $|T_j| - 1$ tasks of T_j . We denote by $\hat{\pi}_j^q$, for all $j \in S$ and $q \in \{1, \dots, |T_j| - 1\}$, the component of $\hat{\boldsymbol{\pi}}$ associated with the task q covered by $\mathbf{A}_j$. Similarly, we denote by α_j^q , for all $j \in S$ and $q \in \{1, \dots, |T_j|\}$, the component of $\boldsymbol{\alpha}$ associated with the task q covered by $\mathbf{A}_j$. To determine values for the α_j^q variables, we solve the following linear system of equations :

$$\sum_{i=1}^{|T_j|} \alpha_j^i = c_j, \quad \forall j \in S, \quad (\text{A.6})$$

$$\sum_{i=1}^q \alpha_j^i = \hat{\pi}_j^q, \quad \forall j \in S, q \in \{1, \dots, |T_j| - 1\} \quad (\text{A.7})$$

which can be decomposed by column index $j \in S$. Conditions (A.6) ensure that the columns in the current solution of the SPP have a zero reduced cost and all compatible columns in the RP have, thus, a nonnegative reduced cost. Furthermore, conditions (A.7) ensure that the least value among all weighted reduced costs $\bar{c}_j/w_j = (c_j - \boldsymbol{\alpha}^\top \mathbf{A}_j)/w_j$, $j \in I_S$, of the incompatible columns considered in the CP is maximized. A dual solution $\boldsymbol{\alpha} \in \mathbb{R}^m$ that satisfies conditions (A.6) is said to *correspond to the current solution S*.

Finally, observe that, if $(\hat{\boldsymbol{\pi}}, \hat{\sigma}) \in \mathbb{R}^{m-|S|} \times \mathbb{R}$ is an optimal dual solution to the relaxed CP (A.1)–(A.4), then $(\boldsymbol{\alpha}, \hat{\sigma}) \in \mathbb{R}^m \times \mathbb{R}$ forms an optimal solution to the relaxed CP (4.7)–(4.10) when $\boldsymbol{\alpha}$ is computed by solving the equation system (A.6)–(A.7). In these solutions, $\hat{\sigma}$ is the dual value associated with constraints (4.9) and (A.3), which is equal to the optimal value of the relaxed CP.

ANNEXE B

Cette annexe est associée à l'article présenté dans le chapitre 5.

B.1 Detailed sensitivity results

Table B.1 reports the sensitivity results for each tested instance and each of the four algorithms I^2CG_2 , I^2CG_3 , DH, and RMH. The meaning of the columns is the same as for Table 5.5. For each instance, the best gap and the least computational time are highlighted in bold. Dash (-) entries indicate that the algorithm was unable to find a feasible solution.

Tableau B.1 Sensitivity analysis results of I^2CG_2 , I^2CG_3 , DH and RMH.

Instance		I^2CG_2					I^2CG_3					DH					RMH								
		Time	Gap	Iter.	IntS	Sc.	Cols	Time	Gap	Iter.	IntS	Sc.	Cols	Time	Gap	Iter.	IntS	Sc.	Cols						
727	239	1	9	0.25	23	87	4	20538	8	0.25	22	66	4	20574	11	0.38	76	4	8402	4	0.50	20	4	4	8051
		2	9	0.36	23	73	33	23413	8	0.35	20	66	42	19848	9	0.35	59	33	9474	5	0.42	21	5	33	8674
		3	58	0.99	24	110	76	42243	15	0.73	25	93	76	28620	13	0.89	84	76	10036	14	2.30	24	12	76	8541
		4	15	0.56	23	68	61	28957	13	0.18	20	79	61	19614	12	0.63	75	80	9658	5	0.50	22	5	76	8718
		5	62	0.87	29	100	80	51434	21	0.79	20	64	80	24361	13	1.21	87	76	10543	9	1.65	22	7	76	8771
09	348	1	13	0.16	16	47	0	25150	10	0.16	17	43	4	25589	5	0.36	26	0	10735	4	0.16	20	3	4	10709
		2	13	0.18	14	56	9	24408	13	0.16	19	56	9	28438	5	0.36	26	9	10695	4	0.20	17	5	9	10645
		3	45	0.84	19	44	28	44863	17	0.49	17	59	42	28462	13	0.83	64	38	12475	11	1.09	20	8	33	11553
		4	46	1.09	22	56	61	82220	16	1.05	15	55	61	29510	26	1.06	133	80	16582	11	1.18	18	11	47	11682
		5	42	1.29	21	51	80	70734	54	1.04	20	80	71	42321	29	1.27	124	90	16127	60	1.74	25	14	80	13604
94	424	1	26	0.23	19	89	38	51416	31	0.12	16	106	38	45874	36	0.18	68	33	17519	12	0.14	21	8	33	17010
		2	44	0.21	19	93	66	50823	36	0.19	18	101	61	57028	38	0.45	71	66	19615	15	0.26	23	5	71	18184
		3	86	0.12	26	104	80	71340	40	0.06	18	117	80	54589	36	0.07	68	80	20747	19	0.12	29	8	80	19976
		4	175	0.77	46	115	85	322376	87	0.59	26	111	90	86525	133	0.68	278	90	35317	77	0.88	34	16	85	21198
		5	241	1.87	41	111	85	237759	99	0.64	26	131	80	74090	96	1.07	210	80	31193	133	0.93	28	16	80	21655
95	1255	1	2640	0.17	25	429	19	424760	1374	0.14	22	469	14	412841	8276	0.32	959	14	164276	7582	0.82	49	43	42	97485
		2	2411	0.17	24	455	42	457784	1669	0.17	23	489	42	419540	8052	0.22	903	52	171415	7640	1.19	53	30	52	103250
		3	2833	0.23	29	516	66	508050	1893	0.23	24	468	57	467351	9062	1.74	1197	61	192789	7668	2.92	54	100	71	109460
		4	1748	0.32	26	482	71	496406	1959	0.21	25	486	66	464309	9222	0.44	1088	66	192304	7715	-	53	-	-	111408
		5	1750	0.61	25	548	80	550986	3312	0.22	31	456	80	608535	10674	0.53	1210	80	205366	7864	-	58	-	-	115258
757	1290	1	885	0.01	17	322	0	328218	777	0.01	16	390	4	353461	3939	0.04	823	0	102596	562	0.15	66	8	9	59148
		2	915	0.01	17	324	4	326725	855	0.01	17	402	4	337999	3479	0.03	756	14	105320	1163	0.16	75	8	19	62791
		3	1042	0.01	19	317	19	332758	827	0.01	17	329	9	323155	3973	0.01	844	28	110579	1528	0.05	65	10	33	60859
		4	1106	0.01	23	370	52	398472	887	0.00	17	378	57	328690	4294	0.04	941	57	122881	7365	0.21	68	11	61	62094
		5	1151	0.01	21	388	61	338615	995	0.01	19	399	76	334044	4577	0.03	1034	76	131420	7362	0.28	67	9	85	65179
319	1293	1	1500	0.12	23	440	28	286057	1286	0.09	22	470	38	286338	4566	0.26	1212	33	130988	7386	0.79	87	18	28	74412
		2	1482	0.17	20	488	42	277853	1542	0.11	23	462	47	283315	4514	0.24	1170	42	133403	7406	2.14	89	19	76	77687
		3	1719	0.20	25	391	57	307200	1661	0.09	25	554	66	303290	4551	0.34	1128	66	132668	7375	1.32	84	25	71	77647
		4	2078	0.36	28	444	71	332187	2047	0.17	26	547	71	331329	5150	0.56	1385	66	142057	7376	2.23	85	22	71	79578
		5	2059	0.79	28	460	76	371484	2114	0.15	27	491	76	332904	5360	1.06	1536	71	147688	7425	-	93	-	-	80754
320	1740	1	1711	0.05	20	506	14	372602	1673	0.04	18	516	19	361084	8400	0.07	1396	19	150659	1068	0.14	112	7	28	89837
		2	2034	0.06	24	438	38	456921	1874	0.05	17	565	33	369921	8126	0.08	1317	38	151183	1144	0.15	103	13	42	91577
		3	1998	0.06	24	445	57	394734	2076	0.03	21	539	52	396851	9122	0.16	1607	66	165260	7818	0.66	106	21	61	92083
		4	1733	0.31	22	493	47	376756	2069	0.03	22	501	66	377518	9773	0.13	1888	76	175724	7822	0.30	109	7	66	93725
		5	2061	0.30	26	463	76	427415	2034	0.03	24	548	76	393818	9965	0.21	2053	76	185809	7870	0.50	117	21	76	94633
Avg.		1044	0.39	24	284	49	254676	954	0.25	21	305	50	230621	3872	0.47	739	52	92957	3414	0.82	55	15	52	52	54223