PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Algorithme du simplexe en nombres entiers avec décomposition
Auteur: Author:	Abdelouahab Zaghrouti
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Zaghrouti, A. (2016). Algorithme du simplexe en nombres entiers avec décomposition [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2176/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/2176/
Directeurs de recherche: Advisors:	Issmaïl El Hallaoui, & François Soumis
Programme: Program:	Mathématiques de l'ingénieur

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

ALGORITHME DU SIMPLEXE EN NOMBRES ENTIERS AVEC DÉCOMPOSITION

ABDELOUAHAB ZAGHROUTI
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
MAI 2016

© Abdelouahab Zaghrouti, 2016.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ALGORITHME DU SIMPLEXE EN NOMBRES ENTIERS AVEC DÉCOMPOSITION

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

M. DESAULNIERS Guy, Ph. D., président
M. EL HALLAOUI Issmaïl, Ph. D., membre et directeur de recherche
M. SOUMIS François, Ph. D., membre et codirecteur de recherche
M. DESROSIERS Jaques, Ph. D., membre
M. VALÉRIO DE CARVALHO José Manuel Vasconcelos, Ph. D., membre externe

DÉDICACE

À l'âme de mon père

REMERCIEMENTS

Mes remerciements les plus profonds vont à M. Issmail El Hallaoui, directeur de recherche, pour son encadrement exceptionnel ainsi que son support continu et inconditionnel.

Je remercie très fortement M. François Soumis, codirecteur de recherche, pour son accompagnement remarquable et son soutien généreux tant moral que financier.

Je remercie enfin tous ceux qui m'ont soutenu tout au long de mes études ainsi que ceux qui ont contribué de près ou de loin à l'accomplissement de ce travail.

RÉSUMÉ

L'objectif général de cette thèse est de développer un algorithme efficace pour la résolution du *problème de partitionnement d'ensemble* (SPP : Set Partitioning Problem). Le SPP est un problème très connu de la programmation linéaire en nombres entiers. Il consiste à partitionner un ensemble de tâches (ex : vols d'avion, segments de trajet d'autobus, ...) en sous-ensembles (routes de véhicules ou suites de tâches effectuées par une personne) de sorte que les sous-ensembles sélectionnés aient un coût total minimum tout en couvrant chaque tâche exactement une et une seule fois.

Le SPP est en général résolu par la méthode *branch-and-price*. Cette méthode peut être excessivement lente dans le cas de problèmes difficiles de très grande taille. Le paradigme derrière la méthode est "dual" ou "tout ou rien" dans le sens où une solution entière du problème est en général obtenue très tard ou à la fin de la résolution pour les problèmes pratiques. Avoir une solution entière rapidement est très apprécié en pratique.

En plus, il est très fréquent, en pratique, de vouloir optimiser un problème pour lequel on connaît déjà une solution avec une bonne information primale que l'on veut, au moins, améliorer. La méthode branch-and-price n'est pas adaptée pour tirer avantage d'une telle situation.

Une approche "primale" est mieux appropriée pour la résolution du SPP (ex : planification d'horaires de chauffeurs d'autobus). L'approche, en question, s'appelle *l'algorithme du simplexe en nombres entiers* et consiste à commencer d'une solution initiale connue et effectuer une série de petites améliorations de façon à produire une suite de solutions présentant des coûts décroissants et convergeant vers une solution optimale.

Plusieurs auteurs ont proposé par le passé des algorithmes pour résoudre le SPP d'une façon primale. Malheureusement, aucun de ces algorithmes n'est assez efficace pour être utilisé en pratique. Le principal facteur derrière cela est la nature fortement dégénérée du SPP. Pour chaque solution, il y a un très grand nombre de bases permettant d'identifier des mouvements vers des solutions voisines. Le phénomène de la dégénérescence implique qu'il est difficile, et même combinatoire, de passer d'une solution entière à une autre ; mais ces algorithmes ne proposent pas de techniques efficaces pour pallier ce phénomène.

Donc, plus précisément, l'objectif de cette thèse est de proposer une implémentation de l'algorithme du simplexe en nombres entiers pratique et efficace contre la dégénérescence. C'est-à-dire que l'implémentation recherchée doit être capable de résoudre des SPPs de grande

taille ; et elle doit aussi être en mesure d'exploiter une solution initiale donnée et produire, itérativement et dans des temps raisonnablement courts, des solutions améliorées.

Pour ce faire, nous commençons, dans un premier travail, par l'exploitation des idées d'un algorithme appelé *simplexe primal amélioré* (IPS : Improved Primal Simplex). Cet algorithme cerne efficacement le phénomène de la dégénérescence lors de la résolution d'un programme linéaire quelconque. Ainsi, nous proposons un algorithme inspiré par IPS et adapté au contexte du SPP (nombres entiers).

L'algorithme, baptisé *simplexe en nombres entiers avec décomposition*, commence à partir d'une solution initiale avec une bonne information primale. Comme dans IPS, il améliore itérativement la solution courante en décomposant le problème original en deux sous-problèmes : un premier sous-problème, appelé *problème réduit*, qui est un petit SPP, permet d'améliorer la solution en ne considérant que les colonnes dites *compatibles* avec la solution courante. Un deuxième sous-problème, appelé *problème complémentaire*, ne considérant que les colonnes *incompatibles* avec la solution courante, permet de trouver une direction de descente combinant plusieurs variables qui garantit d'avoir une meilleure solution, mais pas nécessairement entière.

Le domaine réalisable du problème complémentaire, relaxé de toute contrainte d'intégralité, représente un cône des directions réalisables. Une contrainte supplémentaire, appelée *contrainte de normalisation*, lui est ajoutée pour assurer qu'il soit borné. Les directions qu'il trouve ont la caractéristique d'être *minimales* dans le sens où elles ne contiennent aucune sous-direction réalisable. Cette caractéristique, accompagnée d'une technique de *pricing partiel (partial pricing)* appelée *multi-phase*, fait que, dans la majorité des itérations, le problème complémentaire trouve directement des directions qui mènent vers des solutions entières. Dans le restant des cas, où les directions trouvées mènent vers des solutions fractionnaires, un branchement en profondeur permet souvent d'aboutir rapidement à une solution entière.

Nous avons testé ce nouvel algorithme sur des instances d'horaires de chauffeurs d'autobus ayant 1600 contraintes et 570000 variables. L'algorithme atteint la solution optimale, ou une solution assez proche, pour la majorité de ces instances ; et ceci dans un temps qui représente une fraction de ce qu'aurait demandé un solveur commercial tel que CPLEX ; sachant que ce dernier n'arrive même pas à trouver une première solution réalisable après une durée de plus de 10 heures d'exécution sur certaines instances.

L'algorithme, dans sa première version, représente à notre avis une première implémentation de l'algorithme du simplexe en nombres entiers à être capable de résoudre des instances de SPP de grande taille dans des temps acceptables en pratique. Toutefois, il souffre encore de quelques limitations telles que la nécessité de développer un branchement complexe pour

pouvoir améliorer la qualité des solutions trouvées. Cela est dû au fait que le problème complémentaire présente une structure difficilement exploitable par CPLEX. Une autre limitation de cette implémentation est qu'elle ne permet pas de supporter les contraintes supplémentaires qui ne sont pas de type partitionnement.

Dans un deuxième travail, nous améliorons notre algorithme en généralisant certains aspects de son concept. Notre objectif dans cette étape est d'éviter d'implémenter un branchement complexe et exhaustif tout en permettant à notre algorithme de pouvoir considérer des contraintes supplémentaires.

Nous revoyons donc la façon avec laquelle l'algorithme décompose le problème et nous proposons une méthode de décomposition dynamique où l'intégralité de la solution est contrôlée au niveau du problème réduit au lieu du problème complémentaire. Ainsi, le problème complémentaire n'est plus responsable de trouver une direction menant à une solution entière mais plutôt une direction de descente quelconque ; et c'est le problème réduit qui s'occupe de chercher une solution entière autour de cette direction de descente en déléguant le branchement au solveur commercial.

Avec cette décomposition dynamique, l'algorithme atteint une solution optimale, ou presque optimale, pour toutes les instances, tout en maintenant le même ordre de grandeur des temps d'exécution de la version précédente.

Dans un troisième travail, nous nous donnons l'objectif d'améliorer la performance de l'algorithme. Nous visons de rendre les temps d'exécution de l'algorithme plus rapides sans perdre tous les avantages introduits par le deuxième travail. Nous constatons, alors, que la minimalité des directions de descente exigée par le problème complémentaire est un facteur qui favorise l'intégralité des solutions subséquentes, mais représente, aussi, un élément de ralentissement puisqu'il force l'algorithme à faire plusieurs petits pas, vers des solutions adjacentes uniquement, en direction de sa solution finale.

Nous changeons, alors, le modèle du problème complémentaire pour lui permettre de trouver des directions de descente non minimales. Le nouveau modèle arrive, ainsi, à aller directement vers des solutions entières non adjacentes présentant des améliorations considérables dans le coût ; et ceci en un nombre d'itérations très réduit qui ne dépasse pas deux itérations pour les instances de grande taille dans nos tests. Une solution optimale est toujours atteinte et le temps global d'exécution est réduit par au moins un facteur de cinq sur toutes les instances. Ce facteur est de l'ordre de dix pour les instances de grande taille.

Avec ces trois travaux, nous pensons avoir proposé un algorithme du simplexe en nombres entiers efficace qui produit des solutions de qualité en des temps courts.

ABSTRACT

The general objective of this thesis is to develop an efficient algorithm for solving the *Set Partitioning Problem* (SPP). SPP is a well known problem of integer programming. Its goal is to partition a set of tasks (e.g. plane flights, bus trip segments, ...) into subsets (vehicle routes or set of tasks performed by a person) such that the selected subsets have a minimum total cost while covering each task exactly once.

SPP is usually solved by the method of *branch-and-price*. This method can be excessively slow when solving difficult problems of large size. The paradigm behind the method is "dual" or "all or nothing" in the sense that an integer solution of the problem is generally obtained very late or at the end of the solution process for large instances. In practice, having an integer solution quickly is very appreciated.

Also, it is very common in practice to solve a problem for which a solution having good primal information is already known. We want to, at least, improve that solution. The branch-and-price method is not suitable to take advantage of such a situation.

A "primal" approach fits better for the solution of the SPP (e.g. bus driver scheduling). The approach is called the *Integral Simplex algorithm*. It consists of starting from a known initial solution and performing a series of small improvements so as to produce a series of solutions with decreasing costs and converging towards an optimal solution.

Several authors have, in the past, proposed algorithms for solving the SPP in using a primal paradigm. Unfortunately, none of these algorithms is effective enough to be used in practice. The main factor behind this is the highly degenerate nature of the SPP. For each solution, there is a very large number of bases that permit to identify transitions to neighbor solutions. The degeneracy implies that it is difficult, and even combinatorial, to go from an integer solution to another; but these algorithms do not offer effective techniques to overcome degeneracy.

So, specifically, the aim of this thesis is to introduce an implementation of the Integral Simplex that is effective against degeneracy in practice. This means that the intended implementation must be able to solve SPPs of large size; and it must also be able to benefit from a given initial solution and produce, iteratively and in reasonably short time, improved solutions.

To do this, we first use ideas from an algorithm called *Improved Primal Simplex* (IPS) algorithm. This algorithm helps the primal simplex algorithm in effectively coping with degeneracy when solving linear programs. Thus, we propose an algorithm inspired by IPS

and adapted to the context of the SPP.

The algorithm, called *Integral Simplex Using Decomposition*, starts from an initial solution with good primal information. As in IPS, it iteratively improves the current solution by decomposing the original problem into two sub-problems: a first sub-problem, called *reduced problem*, which is a small completely non-degenerate SPP that improves the solution by considering only the columns that are said to be *compatible* with the current solution. A second sub-problem, called *complementary problem*, considers only the columns that are *incompatible* with the current solution. The complementary problem finds a descent direction, combining several variables, that guarantees to have a better solution; but not necessarily integer.

The feasible domain of the complementary problem, where all the integrality constraints are relaxed, is a cone of feasible directions. An additional constraint, called *normalization constraint*, is added to ensure that the problem is bounded. The directions found are *minimal* in the sense that they do not contain any feasible sub-direction. This minimality feature, combined with a partial pricing technique called *multi-phase*, helps the complementary problem in finding directions that directly lead to integer solutions in the majority of iterations. In the remaining cases, where the directions lead to fractional solutions, a quick deep branching often lead to an integer solution.

We tested the new algorithm on bus driver scheduling problems having 1600 rows and 570000 columns. The algorithm reaches an optimal, or near optimal, solution for the majority of these problems; solution times represent a fraction of what would have taken a commercial solver such as CPLEX. The latter does not even find a first feasible solution within a 10 hour runtime period for some of those problems.

We think that the algorithm, under its first version, is a first implementation of the integral simplex method that was able to solve large SPP problems within acceptable times in practice. However, it still has some limitations such as the need to develop a complex branching to improve the quality of the solutions found. This is due to the fact that the complementary problem presents a structure that is not suitable to handle. Another limitation of this implementation is the fact that it does not consider supplementary non partitioning constraints.

In a second paper, we improve our algorithm generalizing certain aspects of its concept. Our goal in this step is to avoid implementing a complex and exhaustive branching while allowing our algorithm to consider supplementary constraints.

We review, therefore, the way in which the algorithm decomposes the problem and propose a

method of dynamic decomposition where the integrality of the solution is controlled within the reduced problem instead of the complementary problem. Thus, the complementary problem is no longer responsible for finding a direction leading to an integer solution but only a descent direction; and the reduced problem handles the integrality of the solution, while searching around this descent direction, by delegating the branching to the commercial solver.

With this dynamic decomposition, the algorithm reaches an optimal or near optimal solution for all instances; while maintaining execution times comparable to the ones from the previous version.

In a third paper, we target the objective of improving the performance of the algorithm. We aim to make the algorithm run faster without losing the benefits introduced by the second paper. We observe, then, that the minimality of descent directions, required by the complementary problem, forces the algorithm to make small steps towards adjacent solutions.

We then change the model of the complementary problem to let it find non-minimal descent directions. The new model is, thus, able to go directly to non-adjacent integer solutions with significant improvements in the cost, in a very limited number of iterations that does not exceed two iterations for large problems in our tests. An optimal solution is always reached and the execution time is reduced by at least a factor of five on all instances. This factor is about ten for large instances.

With these three papers, we believe we have introduced an effective integral simplex algorithm that produces quality solutions in short times.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT viii

TABLE DES MATIÈRES xi

LISTE DES TABLEAUX xiv

LISTE DES FIGURES xv

LISTE DES SIGLES ET ABRÉVIATIONS xvi

CHAPITRE 1 INTRODUCTION 1

CHAPITRE 2 REVUE DE LITTÉRATURE 5
2.1 Propriétés du polyèdre du SPP 5
2.1.1 Quasi-intégralité 5
2.1.2 Séquence monotone de solutions entières 6
2.1.3 Non décomposabilité 6
2.2 Premiers algorithmes du simplexe en nombres entiers 8
2.2.1 Méthode de Balas et Padberg 8
2.2.2 Méthode be base entière 9
2.2.3 Méthode du simplexe en nombres entiers 9
2.3 Méthode du simplexe primal améliorée (IPS : Improved Primal Simplex) . . 10
2.3.1 Décomposition RP-CP 11
2.3.2 Minimalité 12
2.3.3 Cadre général 13
2.4 Travaux parallèles 14

CHAPITRE 3 ORGANISATION DE LA THÈSE 16

CHAPITRE 4 ARTICLE 1 : INTEGRAL SIMPLEX USING DECOMPOSITION FOR

THE SET PARTITIONING PROBLEM 18
4.1 Introduction 19
4.2 Preliminaries 20
4.2.1 Results from Balas and Padberg 20
4.2.2 Results from the improved primal simplex (IPS) 22
4.3 The integral simplex using decomposition algorithm 25
4.3.1 Improvement of the current integer solution by RP 26
4.3.2 Improvement of the current integer solution by CP 28
4.3.3 Branching to obtain column-disjoint solutions to CP 31
4.3.4 Improving the current integer solution by CP is sufficient to reach optimality 32
4.4 Improvement of Balas and Padberg results on the sequence of adjacent bases leading to optimality 35
4.4.1 The type of pivots used by the ISUD algorithm 35
4.4.2 The number of pivots in ISUD 36
4.5 Implementation tips 36
4.6 Experimentation 38
4.6.1 Instances 39
4.6.2 Results 41
4.7 Conclusion 47

CHAPITRE 5 ARTICLE 2 : IMPROVING SET PARTITIONING PROBLEM SOLUTIONS BY ZOOMING AROUND AN IMPROVING DIRECTION 53
5.1 Introduction 54
5.1.1 Literature review 54
5.1.2 Main contributions 56
5.2 RP–CP decomposition 57
5.3 Integral simplex using decomposition (ISUD) 62
5.4 Zooming around an improving direction 63
5.4.1 Zooming algorithm 64
5.4.2 Insights into the zooming algorithm 66
5.5 Experimentation 69
5.5.1 Instances 70
5.5.2 Numerical results and some implementation tips 71
5.6 Possible extensions 78

CHAPITRE 6 ARTICLE 3 : IMPROVED INTEGRAL SIMPLEX USING DECOM-

POSITION FOR THE SET PARTITIONING PROBLEM 81
6.1 Introduction 82
6.2 ISUD 83
6.3 I^{2}SUD, an Improved ISUD 86
6.3.1 Improved model 86
6.3.2 New normalization weights 89
6.3.3 Improved algorithm 91
6.3.4 Remarks on I^{2}SUD 94
6.4 Numerical results 96
6.5 Conclusion 99

CHAPITRE 7 DISCUSSION GÉNÉRALE 100

CHAPITRE 8 CONCLUSION ET RECOMMANDATION 102

RÉFÉRENCES 103

LISTE DES TABLEAUX

Table 4.1	Mono-phase ISUD vs CPLEX (small instance)	42
Table 4.2	Multi-phase ISUD vs CPLEX (small instance)	43
Table 4.3	ISUD results for medium instances	45
Table 4.4	ISUD results for large instances	46
Table 4.5	Information on initial solutions	47
Table 5.1	ISUD vs. ZOOM (nonzeros)	71
Table 5.2	ISUD vs. ZOOM (small instances)	74
Table 5.3	ISUD vs. ZOOM (medium instances)	75
Table 5.4	ISUD vs. ZOOM (large instances)	76
Table 5.5	ISUD CP vs. ZOOM CP (small instances)	77
Table 5.6	ISUD CP vs. ZOOM CP (medium instances)	78
Table 5.7	ISUD CP vs. ZOOM CP (large instances)	79
Table 6.1	Example	93
Table 6.2	ISUD vs. I^{2}SUD (quality and performance)	98
Table 6.3	ISUD vs. I^{2}SUD (iterations and directions)	99

LISTE DES FIGURES

Figure 4.1 Mono-phase ISUD vs Multi-Phase ISUD vs CPLEX (small problem, instance 1, 50%) 44
Figure 4.2 Multi-phase ISUD vs CPLEX (small problem, instance 4, 50%) . . . 44
Figure 5.1 Zooming around an improving direction 67

LISTE DES SIGLES ET ABRÉVIATIONS

SPP	Set Partitioning Problem
IPS	Improved Primal Simplex
RP	Reduced Problem
CP	Complementary Problem
ISUD	Integral Simplex Using Decomposition
I^2SUD	Improved Integral Simplex Using Decomposition

CHAPITRE 1 INTRODUCTION

La résolution de problèmes en nombres entiers est connue comme étant très difficile depuis son introduction il y a plus de cinquante ans. Les chercheurs continuent à étudier la structure discrète de ces problèmes en nombres entiers pour développer des algorithmes efficaces. Et parmi ces problèmes, le problème de partitionnement d'ensemble (SPP : Set Partitioning Problem) présente beaucoup d'intérêt. Ce dernier est très utilisé pour modéliser des problèmes d'optimisation dans différents secteurs industriels. Parmi ses applications, on trouve : horaires de chauffeurs d'autobus (Desrochers & Soumis (1989)), horaires d'équipages d'avions (Barnhart et al. (1998)), tournées de véhicules (Desrochers et al. (1992)), etc.

La formulation de SPP est comme suit :

$$\begin{aligned} & \min \quad c \cdot x && (1.1)\\ (\mathbb{P}) \quad & s.\text{à}. \quad Ax = e && (1.2)\\ & \qquad x_j \quad \text{binaire}, \quad j \in \{1, \ldots, n\} && (1.3) \end{aligned}$$

avec A une matrice binaire $m \times n$, c le vecteur coût, et $e = (1, \ldots, 1)$.

Dans le cas d'horaires de chauffeurs d'autobus, par exemple, le SPP considère un ensemble de tâches à effectuer (segments de trajet d'autobus) et un ensemble de possibilités de grouper ces tâches ensemble (horaires admissibles pour un chauffeur quelconque). À chaque horaire est affecté un coût (salaire du chauffeur pour effectuer le chemin par exemple). Le SPP consiste, alors, à trouver un ensemble de chemins où chaque tâche est couverte une et une seule fois par un chemin de l'ensemble. Un ensemble de chemins satisfaisant ce critère est une *solution* du problème. Parmi toutes les solutions du problème, celles qui présentent un coût total minimal sont appelées *solutions optimales*.

Les contraintes du problème (1.2) qui consistent à couvrir chaque tâche une et une seule fois sont appelées *contraintes de partitionnement*. Une contrainte de partitionnement représente un cas particulier de ce que l'on appelle une *contrainte de réalisabilité* dans un contexte plus général. Aussi, dans une solution, on ne peut pas considérer une fraction de chemin couvrant une tâche donnée (plusieurs chauffeurs effectuant le même segment de trajet). On dit que la solution respecte les *contraintes d'intégralité* (1.3) et est *entière*.

Le SPP peut inclure d'autres contraintes, appelées *contraintes supplémentaires*, qui ne sont pas des contraintes de partitionnement. Ces contraintes permettent de considérer d'autres aspects dans la réalisabilité des solutions (ensembles de chemins) trouvées tels que le nombre

maximum de chauffeurs disponibles. Souvent, le nombre de contraintes supplémentaires représente un faible pourcentage du nombre total de contraintes du problème. Un problème de partitionnement dont toutes les contraintes sont des contraintes de partitionnement est appelé un *problème de partitionnement pur* ou simplement SPP.

Les problèmes que l'on trouve en industrie contiennent souvent des contraintes de non partitionnement inhérentes aux règles de gestion. En plus, ces problèmes présentent un grand nombre de tâches à couvrir. Le nombre de chemins possibles devient, par conséquent, très élevé et presque impossible à énumérer en totalité. C'est pourquoi les problèmes de partitionnement sont, en général, résolus par un algorithme de *génération de colonnes* où les chemins possibles ne sont pas connus a priori, mais sont construits par un problème auxiliaire, à la demande et au fur et à mesure, le long du processus de résolution.

Le modèle du SPP (pur) continue, cependant, à faire l'objet d'études de recherche sur le plan théorique. En effet, d'une part, ce modèle est utilisé comme point de départ pour plusieurs variantes non pures du problème. D'autre part, en pratique, les contraintes supplémentaires sont souvent des contraintes "soft" (molles : qui peuvent être transférées dans l'objectif) et on se retrouve avec un modèle pur. Ainsi, le modèle est simple et présente des caractéristiques théoriques intéressantes.

La littérature sur le SPP est abondante. En général, un algorithme qui résout le SPP est caractérisé par la suite de solutions qu'il trouve avant de converger vers une solution optimale. Principalement, il y a trois familles d'algorithmes (Letchford & Lodi (2002)) :

- algorithmes *duaux fractionnaires* : les solutions parcourues satisfont les contraintes de réalisabilité et d'optimalité (coût minimum). La solution optimale est trouvée quand les contraintes d'intégralité sont satisfaites. La méthode populaire *branch-and-bound* fait partie de cette catégorie (la borne inférieure est améliorée par branchement ou coupes).
- algorithmes *duaux entiers* : les solutions parcourues respectent les contraintes d'intégralité et d'optimalité. Seule une solution qui satisfait les contraintes de réalisabilité est réalisable optimale et permet à l'algorithme de s'arrêter. Ce genre d'algorithme n'est pas très utilisé en pratique. L'algorithme de Gomory (1963) en est le seul exemple connu.
- algorithmes *primaux* : les solutions intermédiaires sont toutes réalisables et entières (respectent les contraintes d'intégralité). Le coût est amélioré d'une solution à une autre jusqu'à atteindre une solution ayant un coût minimal et est donc optimale. Un exemple est l'algorithme de Balas & Padberg (1975).

En pratique, un algorithme très utilisé pour résoudre le SPP est le *branch-and-cut* qui combine

la méthode d'énumération implicite (branch-and-bound) avec la méthode des *plans coupants* pour ajouter des coupes au niveau des nœuds de branchement. Un autre algorithme est le *branch-and-price*. Celui-ci combine le branch-and-bound avec la génération de colonnes pour pouvoir résoudre des problèmes de grande taille (nœuds de l'arbre de branchement sont résolus avec la génération de colonnes). Quand on combine, à la fois, les coupes et la génération de colonnes avec le branch-and-bound, on parle de *branch-price-and-cut*.

La nature combinatoire globale de l'algorithme branch-and-cut (arbre de branchement du branch-and-bound) fait que son temps d'exécution peut augmenter exponentiellement avec la taille du problème malgré toutes les simplifications dont il peut bénéficier (coupe de branches de l'arbre de branchement ou ajout de coupes). Or, la taille des problèmes à résoudre continue à augmenter avec les besoins de l'industrie. Par exemple, les flottes d'avions deviennent plus grandes à cause des fusions et des alliances des compagnies aériennes ; et les périodes ou horizons de planification deviennent plus longues. L'algorithme peut donc présenter des temps d'exécution très longs qui peuvent atteindre plusieurs jours. Aussi, l'aspect "dual" de cette approche fait que l'algorithme peut, dans des cas difficiles, consommer tout le temps maximal que l'on se permet de lui allouer sans trouver une première solution entière.

De plus, les cas de ré-optimisation de problèmes sont fréquents en pratique (exemple : annulation de vols lors d'une tempête hivernale). Dans ces cas, une solution optimale est connue mais certaines conditions ont changé et il faut donc résoudre le problème de nouveau ; le branch-and-cut ne tire pas grand profit de l'ancienne solution optimale déjà connue.

À tous ces éléments sus-mentionnés, s'ajoute le fait que l'amélioration des temps d'exécution est toujours un besoin incessant. Il est donc important de trouver une méthode alternative plus efficace pour la résolution du SPP.

Parmi les critiques attribuées au branch-and-cut, on pense que son approche duale est le facteur principal qui limite sa performance. Une méthode alternative plus efficace devrait donc être basée sur un paradigme primal. En fait, une telle méthode existe ! C'est prouvé par Balas & Padberg (1972).

En effet, le SPP possède une structure particulièrement simple. Cette structure particulière, appelée *quasi-intégralité*, a été découverte par Trubin (1969). La propriété de cette structure est que deux solutions adjacentes pour SPP sont aussi adjacentes pour SPP relaxé (de ses contraintes d'intégralité). À partir de cette propriété, Balas & Padberg (1972) font remarquer que l'on peut appliquer la *méthode du simplexe*, faisant abstraction des contraintes d'intégralité, sur le SPP. On peut, ainsi, théoriquement, partir d'une solution initiale et atteindre une solution optimale en passant par une suite de solutions adjacentes où le coût décroît avec chaque solution jusqu'à sa valeur optimale.

Cette approche est connue sous le nom du "*simplexe en nombres entiers*". Des travaux subséquents ont proposé des algorithmes dans le cadre de cette approche (e.g. Balas & Padberg (1975)). La majorité de ces algorithmes ne sont pas très efficaces parce que, d'une part, les SPPs sont des problèmes fortement dégénérés. Pour chaque solution, il y a un très grand nombre de bases permettant d'identifier des mouvements vers des solutions voisines. Toutefois la plupart de ces bases ne permettent que des pivots dégénérés qui n'améliorent pas la solution. D'autre part, ces algorithmes proposent des techniques énumératives combinatoires pour contourner la dégénérescence.

Dans le cadre de cette thèse, nous nous donnons l'objectif de proposer un algorithme du *simplexe en nombres entiers* primal qui résout efficacement les SPPs de grande taille et qui soit, plus particulièrement, très utile dans le cadre de problèmes de ré-optimisation.

Nous contribuons, dans le cadre des travaux d'étude effectués pour atteindre cet objectif, avec trois articles de recherche. Sur le plan théorique, nous contribuons avec l'introduction d'une méthode primale qui trouve plus facilement la séquence de solutions de Balas et Padberg en utilisant une approche simple. Cette dernière consiste à décomposer, itérativement, le problème en deux sous-problèmes qui se complètent pour trouver facilement une solution subséquente améliorée dans la séquence de Balas et Padberg.

Nous enrichissons ensuite notre méthode par une généralisation de ce concept de décomposition pour augmenter la performance de notre méthode et d'y ajouter la possibilité de prendre en considération des contraintes supplémentaires.

Enfin, nous introduisons une nouvelle formulation permettant de passer à des solutions non adjacentes dans la séquence de Balas et Padberg. Ceci permet de réduire le nombre de solutions que traverse la méthode avant d'arriver à la fin de la séquence de Balas et Padberg, soit la solution optimale.

Sur le plan pratique, nous considérons que la méthode que nous introduisons est très compétitive avec les méthodes les plus utilisées en industrie qui sont déjà très matures. Nous effectuons nos tests sur des instances de rotations d'équipages aériens (pairing) ainsi que des instances de grande taille de problèmes d'horaires de chauffeurs d'autobus (1600 tâches et 570000 chemins). Avec notre méthode, nous arrivons à atteindre une solution optimale en une fraction du temps que demanderait une approche classique.

Les trois articles de cette thèse font l'objet des chapitres 4, 5 et 6. Le chapitre 3 donne une brève description de chaque article.

CHAPITRE 2 REVUE DE LITTÉRATURE

Selon la classification introduite par Letchford & Lodi (2002) et mentionnée dans le chapitre précédent, l'approche proposée dans cette thèse fait partie de la famille des algorithmes primaux. Pour cette raison, nous consacrons notre revue de littérature à la résolution primale du SPP et principalement au paradigme du simplexe en nombres entiers.

Dans la section 2.1, nous présentons des caractéristiques théoriques du SPP qui sont derrière l'idée du simplexe en nombres entiers. Dans la section 2.2, nous passons en revue les algorithmes existants qui souffrent tous de la dégénérescence. Dans la section 2.3, nous décrivons l'algorithme du simplexe primal amélioré (IPS) qui est efficace contre la dégénérescence et dont le fondement théorique constitue la base de notre travail. Enfin, dans la section 2.4, nous mentionnons quelques travaux parallèles avec lesquels nous partageons des éléments de notre objectif.

2.1 Propriétés du polyèdre du SPP

Nous commençons notre revue de littérature par une revue des caractéristiques intéressantes que présente le polyèdre du SPP : la quasi-intégralité, l'existence d'une séquence monotone de solutions entières menant à une solution optimale et la non décomposabilité de l'ensemble de pivots à effectuer pour passer d'une solution entière à une autre voisine.

Considérons la formulation du SPP (1.1)-(1.3). Notons $(\mathbb{P}')$ la relaxation linéaire de $(\mathbb{P})$ obtenu en remplaçant x_j binaire par $x_j \geq 0$, $j \in \{1, \ldots, n\}$. Soit $\bar{A} = B^{-1}A = (\bar{a}_{ij})$ où B est une base, $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$.

2.1.1 Quasi-intégralité

L'histoire de l'algorithme du simplexe en nombres entiers pour le SPP commence avec Trubin (1969) quand il observe que le polytope du problème relaxé $(\mathbb{P}')$ est "quasi entier" ; c'est-à-dire que toute arête de l'enveloppe convexe (conv$(\mathbb{P})$) des solutions de $(\mathbb{P})$ est aussi une arête du polytope de $(\mathbb{P}')$. Cette propriété est très intéressante puisqu'elle implique l'existence d'un chemin dans le polytope de $(\mathbb{P}')$, ne contenant que des sommets entiers, entre toute paire de sommets entiers.

Mais, Balas & Padberg (1972) trouvent le résultat de Trubin insuffisant puisque, pour passer d'un sommet entier à un autre, il ne donne ni une borne supérieure sur le nombre de pivots nécessaires, ni une garantie sur l'existence d'un chemin composé de points extrêmes à coûts

décroissants.

Balas & Padberg (1972) alors prouvent un résultat fondamental de cette branche de recherche : l'existence d'une suite de solutions entières ayant des coûts décroissants qui mène à la solution optimale en au plus m pivots.

Vue l'importance de ce résultat, nous donnons ci-après plus de détails sur les théorèmes fondamentaux introduits par Balas et Padberg.

2.1.2 Séquence monotone de solutions entières

Balas & Padberg (1972) distinguent d'abord les définitions suivantes de l'adjacence : deux bases d'un programme linéaire sont dites adjacentes si elles diffèrent par exactement une seule colonne. Deux solutions de base sont dites adjacentes si elles représentent deux sommets adjacents du polytope de ($\mathbb{P}'$). Cette distinction est nécessaire puisque deux bases adjacentes peuvent être associées à une même solution et deux solutions adjacentes peuvent être associées à deux bases non adjacentes.

La solution correspondant à une base B est entière si et seulement s'il existe un sous-ensemble Q de colonnes de B tel que $\sum_{j \in Q} A_j = e$. Si A est de plein rang, toute solution entière pour $\mathbb{P}'$ est une solution de base.

Pour $i = 1, 2$, soit x^i une solution de base entière pour $\mathbb{P}'$, B_i une base associée à x^i, I_i et J_i les ensembles d'indices correspondant aux colonnes en base et hors base, et $Q_i = \{j | x_j = 1\}$.

Théorème 3.1 (Balas & Padberg (1972))
Si x^2 est une solution optimale pour $\mathbb{P}$, alors il existe une séquence de bases adjacentes B_{10}, B_{11}, B_{12}, ..., B_{1p} telle que $B_{10} = B_1, B_{1p} = B_2$, et

a) les solutions de base correspondantes $x^1 = x^{10}, x^{11}, x^{12}, \ldots, x^{1p} = x^2$ sont toutes entières,

b) $c \cdot x^{10} \geq \ldots \geq c \cdot x^{1p}$, et

c) $p = |J_1 \cap Q_2|$.

Dans la démonstration de ce théorème, on remarque que l'on autorise les colonnes ayant des coûts réduits nuls ($\bar{c}_j = 0$) à entrer en base où des pivots sur des $\bar{a}_{ij}$ négatifs peuvent être effectués.

2.1.3 Non décomposabilité

Balas & Padberg (1975) observent qu'il est difficile d'identifier une séquence de pivots améliorante quand la solution x^2 est inconnue. Ils affirment : *"les SPPs ont tendance à être très*

dégénérés. Le polytope réalisable contient souvent un très grand nombre de sommets adjacents à un sommet donné. En outre, toute technique lexicographique ou similaire n'est d'aucune utilité pour faire face à la dégénérescence puisque la séquence de pivots requise pour atteindre un sommet adjacent peut nécessiter des pivots sur des coefficients négatifs dans une ligne dégénérée".

Les théorèmes suivants de Balas & Padberg (1975) établissent les relations entre deux solutions entières, adjacentes ou non, et l'ensemble des colonnes sur lequel il faut effectuer des pivots pour passer d'une solution à l'autre.

Théorèmes 1 et 2 (Balas & Padberg (1975))
Considérons x^1 une solution de base entière et I_1, J_1, et Q_1 les ensembles d'indices correspondants. Il existe $Q \subseteq J_1$ tel que

$$Q^+ = \{k | \sum_{j \in Q} \quad \bar{a}_{kj} = 1\} \quad \subseteq \quad Q_1, \tag{2.1}$$

$$Q^- = \{k | \sum_{j \in Q} \quad \bar{a}_{kj} = -1\} \quad \subseteq \quad I_1 \cap \bar{Q}_1 \tag{2.2}$$

$$\{k | \sum_{j \in Q} \quad \bar{a}_{kj} = 0\} \quad \subseteq \quad I_1 \setminus \{Q^+ \cup Q^-\} \tag{2.3}$$

si et seulement si

$$x_j^2 = \begin{cases} 1 & \text{si } j \in Q \cup Q^- \cup (Q_1 - Q^+) \quad \forall j \in \{1, \ldots, n\} \\ 0 & \text{sinon} \end{cases}$$

est une solution entière obtenue à partir de x^1 en effectuant des pivots sur Q.

Ce théorème donne les conditions, dans le tableau de Tucker, sous lesquelles il existe une solution entière adjacente à la solution de base entière courante. On remarque qu'une variable d'une telle solution entière adjacente peut être une variable hors base qui entre en base et (provient de Q) ou une variable de base dégénérée qui devient non dégénérée (provient de Q^-) ou une variable de base non dégénérée qui reste non dégénérée en base (provient de $Q_1 - Q^+$ et est non nulle à la fois dans x^1 et dans x^2).

Remarque : *Les colonnes correspondant aux variables "entrantes" $Q \cup Q^-$ sont disjointes, c'est-à-dire qu'elles ne couvrent pas les mêmes contraintes. Ces variables remplacent celles "sortantes" provenant de Q^+. Autrement dit, on remplace les colonnes de x^1 dont les indices sont dans Q^+ par les colonnes dont les indices sont dans $Q \cup Q^-$ pour obtenir la solution x^2 améliorée.*

Un ensemble $Q \subseteq J_1$ qui satisfait (2.1)-(2.2) est dit *décomposable* si Q peut être partitionné

en deux sous-ensembles Q^* et Q^{**} satisfaisant (2.1)-(2.3).

Théorème 3 (Balas & Padberg (1975))
Soient x^1 et x^2 deux solutions entières de $\mathbb{P}$, avec $Q = J_1 \cap Q_2$. Alors x^2 est adjacente à x^1 si et seulement si Q est non décomposable.

Soient x^1 et x^2 deux sommets non adjacents dans le domaine réalisable de $\mathbb{P}$ reliés par des pivots dans $Q = J_1 \cap Q_2$. Les points a), b) et c) suivants correspondent aux corollaires 3.2, 3.3, et 3.5 de Balas & Padberg (1975).

a) Il existe une partition de $Q, Q = \cup_{i=1}^{h} Q_i$, telle que Q_i satisfait (2.1)-(2.2) et Q_i est non décomposable.
b) Pour tout $H \subseteq \{1, \ldots, h\}$ les pivots sur $\cup_{i \in H} Q_i$ en partant de x^1 mènent vers une solution entière de $\mathbb{P}$.
c) Toute permutation de $\{1, \ldots, h\}$ définit un chemin de sommets adjacents de x^1 vers x^2.

Balas & Padberg (1975) montrent uniquement l'existence d'une suite d'ensemble Q_i non décomposables menant à une solution optimale, mais n'arrivent pas à la trouver facilement. Dans notre premier travail (voir chapitre 4), nous proposons une méthode constructive efficace pour trouver une telle suite.

2.2 Premiers algorithmes du simplexe en nombres entiers

Les premières tentatives de proposer une méthode primale pour la résolution du SPP étaient enregistrées, avant Trubin, avec Gomory (1963) et Young (1968) qui avaient proposé des coupes permettant de maintenir des pivots entiers ; mais, en plus du nombre de pivots nécessaires qui était grand, des problèmes de stabilité numérique se présentaient. Nous présentons, ci-après, les méthodes qui ont suivi et qui se sont données l'objectif de résoudre le SPP dans le cadre du paradigme du simplexe en nombres entiers.

2.2.1 Méthode de Balas et Padberg

Balas & Padberg (1975) proposent une façon pour chercher le prochain terme de leur séquence. Ils décrivent une procédure pour générer tous les sommets entiers adjacents à un sommet donné. Ils remarquent, cependant, que la dégénérescence est le problème principal que rencontre leur algorithme.

En fait, Balas et Padberg effectuent un raisonnement complètement basé sur les éléments du tableau du simplexe pour caractériser la transformation à effectuer sur une solution entière courante pour obtenir une autre solution entière adjacente. Ils proposent un algorithme énu-

mératif qui cherche des ensembles non décomposables Q vérifiant les conditions (2.1)-(2.3). L'algorithme produit, à partir du tableau du simplexe de la solution courante, un autre tableau du simplexe avec de nouvelles colonnes construites artificiellement. Dans ce nouveau tableau, toutes les solutions entières adjacentes sont obtenues en n'effectuant qu'un seul pivot. Pour ce faire, beaucoup de nouvelles colonnes sont, de manière itérative, ajoutées au tableau. À chaque itération, une colonne, à supprimer, est sélectionnée. Plusieurs colonnes du tableau sont combinées, en paire, avec cette colonne sélectionnée et ajoutées au tableau sous forme de nouvelles colonnes. Ainsi, ces ensembles non décomposables recherchés sont bâtis en combinant, à chaque fois, deux colonnes. Il est clair que cette approche est très combinatoire et le nombre de colonnes ajoutées explose avec la taille du problème.

2.2.2 Méthode be base entière

Haus et al. (2001) propose une méthode pour le cas général de problèmes en nombres entiers. Ils l'appellent "méthode de base entière". Haus et al. (2001) appliquent leur algorithme sur le SPP. Leur algorithme consiste à remplacer une variable entrante par une combinaison positive de variables hors base en énumérant toutes les solutions "irréductibles" (qui ne s'écrivent pas sous forme de combinaison linéaire d'autres solutions) d'un problème auxiliaire. Ce dernier cherche, parmi les variables hors base, une variable qui peut passer à 1 sans rendre le système non réalisable. S'il n'en trouve pas, l'algorithme choisit une variable, qui ne peut être "pivotée intégralement" (qui ne peut pas prendre la valeur 1), et la remplace par plusieurs nouvelles colonnes représentant les solutions énumérées d'un systèmes d'inéquations en nombres entiers. Haus et al. (2001) remarquent que leur algorithme doit ajouter un grand nombre de solutions auxiliaires, c'est-à-dire que, de même que pour l'algorithme de Balas et Padberg, beaucoup de colonnes sont ajoutées au problème.

2.2.3 Méthode du simplexe en nombres entiers

Thompson (2002) propose une technique qu'il appelle méthode du simplexe en nombres entiers (Integral Simplex Method). Cette méthode semble un peu plus intéressante que les précédentes. Elle consiste à effectuer des pivots-sur-1 tant que cela est possible. Ce type de pivots permet d'aller vers une solution entière meilleure. On atteint alors une "optimalité locale" et cette première partie de la méthode est appelée "méthode locale". Une deuxième partie, appelée "méthode globale", construit un sous-problème pour chaque colonne ayant un coût réduit négatif et l'ajoute à un arbre (de branchement) de sous-problèmes. Pour chaque sous-problème, la variable choisie est fixée à 1 et les variables précédemment choisies (dans l'ordre de création des sous-problèmes issus du même nœud père) sont fixées à 0. À

chaque itération, un sous-problème est retiré de l'arbre et résolu avec la méthode locale. La solution optimale du problème est la meilleure rencontrée après exploration de l'arbre. Les sous-problèmes deviennent de plus en plus faciles à résoudre avec la profondeur de l'arbre, mais ce dernier peut avoir une très grande taille.

Saxena (2003) démontre que l'optimalité peut être atteinte, avec la méthode de Thompson, en n'utilisant que des pivots-sur-1. Il propose des améliorations à cette méthode en autorisant des pivots-sur-(-1) et en ajoutant des règles d'anti-cyclage. Il permet, aussi, de réduire le nombre de variables hors base ayant des coûts réduits négatifs (puisque ce nombre influence la taille de l'arbre de branchement) en faisant appel à des heuristiques à base d'algorithmes gloutons et de coupes.

Rönnberg & Larsson (2009) proposent une variante de la technique de génération de colonnes adaptée à la méthode du simplexe en nombres entiers de Thompson. Dans le processus d'énumération implicite (branchement) de la "méthode globale", la "méthode locale" résout les nœuds de l'arbre de branchement par génération de colonnes.

2.3 Méthode du simplexe primal améliorée (IPS : Improved Primal Simplex)

Tous les algorithmes, mentionnés dans la section précédente, souffrent de difficultés à résoudre efficacement le SPP. Le facteur principal derrière ces difficultés, tel que énoncé par Balas et Padberg, est la dégénérescence.

L'algorithme du simplexe en nombres entiers cherche à résoudre le SPP par la méthode du simplexe, sauf qu'un choix judicieux d'une séquence de pivots menant à une solution entière reste un défi important à surmonter. La dégénérescence rend ce choix difficile à faire. Donc, à notre avis, toute approche du simplexe en nombres entiers susceptible de résoudre efficacement le SPP doit contrer le phénomène de dégénérescence dans son cadre général.

Dans cette optique, nous avons bâti notre premier travail (article au chapitre 4) sur les développements théoriques effectués par Elhallaoui et al. (2011) pour réduire la dégénérescence dans le cadre général de la résolution, par la méthode du simplexe primal, d'un programme linéaire quelconque.

En fait, en suivant une direction de recherche complètement différente, Elhallaoui et al. (2011) établissent des résultats similaires à ceux de Balas & Padberg (1972, 1975), mais dans le cas d'un programme linéaire quelconque. Ici aussi, comme nous l'avons fait ci-haut avec Balas & Padberg (1972, 1975), nous présentons plus de détails concernant les théorèmes fondamentaux introduits par Elhallaoui et al. (2011). Ainsi, nous mettons en évidence les concepts communs entre les deux directions de recherche.

Nous présentons les résultats de Elhallaoui et al. (2011) en utilisant la notation introduite par Balas & Padberg (1972, 1975) et Raymond et al. (2010a). Considérons un problème linéaire quelconque : min $c \cdot x$ sujet à $Ax = b$, $x \geq 0$. Il faut noter que I et J utilisés ici pour illustrer IPS sont différents de I et J utilisés par Balas et Padberg. Si $x \in R^n$ et $I \subseteq \{1, \ldots, n\}$ est un ensemble d'indices, x_I désigne le sous-vecteur de x indexé dans I. De même, A_I désigne la matrice de dimension $m \times |I|$ dont les colonnes sont indexées dans I. Si $J = \{1, \ldots, n\} \backslash I, \quad x = (x_I, x_J)$ est utilisé même si les indices dans I et J peuvent ne pas être dans l'ordre. De la même manière, les indices en exposant désignent un sous-ensemble de lignes.

2.3.1 Décomposition RP-CP

IPS permet de réduire la difficulté d'un problème dégénéré en le décomposant en deux sous-problèmes : un problème réduit contenant uniquement les colonnes dites *compatibles* avec la solution, comme défini ci-après, et un problème complémentaire contenant les colonnes dites *incompatibles*.

IPS définit le problème réduit (RP : *Reduced Problem*) où les contraintes sont toutes non dégénérées. Ce problème est défini par rapport à une base réalisable B (qui donne une solution de base $\bar{x} = B^{-1}b$).

Soit P l'ensemble d'indices des p variables positives de la solution. La plupart du temps, une solution de base pour un SPP est dégénérée et $p < m$. Soit N l'ensemble d'indices des variables nulles dans la base. En partitionnant les contraintes en (P, N) et en multipliant par B^{-1}, le système linéaire $Ax = b$ devient

$$\begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} \begin{bmatrix} x^P \\ x^N \end{bmatrix} = \begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix}, \tag{2.4}$$

avec $\bar{b}^N = 0$, $\bar{b}^P > 0$, et $\bar{A}^N_j = 0$, $\forall j \in P$.

La j^{eme} variable x_j de $\mathbb{P}'$ est appelée *compatible* avec cette base si et seulement si $\bar{A}^N_j = 0$. Sinon, elle est appelée *incompatible*.

Il est important de souligner, comme il a été remarqué dans Metrane et al. (2010), que d'une façon équivalente une variable est dite compatible avec une base si et seulement si elle est linéairement dépendante des variables non dégénérées (strictement positives) de cette base.

Soient $C \subseteq \{1, \ldots, n\}$ l'ensemble des indices des variables compatibles et $I = \{1, \ldots, n\} \backslash C$ l'ensemble des indices des variables incompatibles. x_C et x_I sont les sous-vecteurs de x contenant, respectivement, les variables compatibles et incompatibles. Le vecteur coût est parti-

tionné en c_C et c_I et la matrice A en A_C et A_I. Le système linéaire (2.4) peut être réécrit comme suit :

$$\bar{A}_C^P x_C + \bar{A}_I^P x_I = \bar{b}^P \tag{2.5}$$
$$\bar{A}_C^N x_C + \bar{A}_I^N x_I = 0 \tag{2.6}$$

RP est obtenu, en imposant $x_I = 0$, comme suit :

$$\text{(RP)} \quad \min_{x_c} \quad c_C \cdot x_C \quad \text{sujet à} \quad \bar{A}_C^P x_C = \bar{b}^P, \quad x_C \geq 0$$

Quand $x_I = 0$, les contraintes indexées par N (2.6) sont satisfaites parce que $\bar{A}_C^N = 0$.

Observations :

- Une variable compatible ayant un coût réduit négatif entre dans la base en effectuant un pivot non dégénéré et une solution de moindre coût est obtenue.
- Une solution optimale x_C^* pour RP peut être facilement obtenue avec la méthode du simplexe primal. En effet, chaque pivot non dégénéré fait avancer la méthode du simplexe primal d'un point extrême vers un autre point extrême de meilleur coût. L'ensemble P peut être réduit si des contraintes dégénérées apparaissent.
- $x = (x_C^*, 0)$ est une solution de $\mathbb{P}'$ meilleure que la solution précédente.

Le problème complémentaire (CP : *Complementary Problem*) est défini quand RP est optimal et P, N, C, et I sont réajustés si nécessaire.

CP est défini par les contraintes dégénérées, les variables incompatibles et le vecteur de coûts réduits $\bar{c}_I$ de la manière suivante :

$$(CP) \quad Z^{CP} = \min_{v} \quad \bar{c}_I \cdot v \quad \text{sujet à} \quad \bar{A}_I^N v = 0,\ e \cdot v = 1,\ v \geq 0$$

où $\bar{c}_I = c_I - c_P \bar{A}_I^P$.

Elhallaoui et al. (2011) ont montré que si CP est non réalisable ou $Z^{CP} \geq 0$ alors $(x_C^*, 0)$ est une solution optimale de $\mathbb{P}'$. De plus, Metrane et al. (2010) ont montré que RP-CP est une décomposition de Dantzig-Wolfe.

2.3.2 Minimalité

Balas et Padberg ont introduit la notion de non décomposabilité de l'ensemble Q comme condition sur l'adjacence entre deux sommets entiers dans le polyèdre de $\mathbb{P}'$. Elhallaoui et al. (2011) remarquent que l'adjacence de deux sommets quelconques est caractérisée par la mi-

nimalité de l'ensemble de pivots à effectuer pour passer d'un sommet à l'autre.

Elhallaoui et al. (2011) ont montré que si CP est réalisable, v^* une solution optimale avec $Z^{CP} < 0$, et S l'ensemble des indices pour lesquels $v_j^* > 0$, alors, en ajoutant les variables indexées dans S à RP, on réduit le coût de la solution et on passe au sommet adjacent. Ils ont montré que S est minimal dans un sens similaire à la non décomposabilité de l'ensemble Q de Balas et Padberg.

Proposition 3 (Elhallaoui et al. (2011))
La combinaison convexe $w = \sum_{j \in S} A_j v_j^*$ est compatible avec RP et l'ensemble S est *minimal* dans le sens où aucune combinaison convexe d'un sous-ensemble strict de S n'est compatible.

2.3.3 Cadre général

L'élément central dans l'efficacité d'IPS contre la dégénérescence est sa décomposition. Nous mentionnons, ci-après, un cadre général de cette décomposition.

Décomposition vectorielle. D'un point de vue "primal", CP cherche une combinaison convexe à faire entrer en base pour améliorer strictement le coût de la solution courante. D'un point de vue "dual", ce problème cherche une solution qui maximise le minimum des coûts réduits des variables hors base incompatibles selon sa décomposition. Gauthier et al. (2015a) formalisent un cadre général pour la décomposition des programmes linéaires dégénérés pour lequel IPS devient un cas particulier. La décomposition générale proposée regarde la décomposition anti-dégénérescence d'un point de vue "dual". Elle consiste à chercher une direction donnant un coût réduit minimal en subdivisant l'ensemble des variables duales en deux sous-ensembles : une partie des variables est fixée et l'autre partie est sujette à l'optimisation.

Ainsi, plusieurs algorithmes connus représentent différentes variantes de cette décomposition générale. Trois cas particuliers sont : la décomposition du simplexe primal fixe toutes les variables duales et choisit une seule variable entrante donnant lieu à des pivots dégénérés ; l'algorithme du Minimum Mean Cycle Canceling, fortement polynomial pour les problèmes de flot, ne fixe aucune variable duale ; et IPS fixe les variables de base non dégénérées et optimisent sur le reste.

Gauthier et al. (2013) recense les trois outils les plus récents pour pallier la dégénérescence primale dans la programmation linéaire : IPS décrit ci-dessus, l'agrégation dynamique de contraintes (DCA : Dynamic Constraint Aggregation) et la règle du Positive Edge. Ci-après, nous donnons une brève description des deux derniers outils.

Agrégation dynamique de contraintes. L'idée de l'agrégation dynamique de contraintes

pour le problème de partitionnement, décrite dans Elhallaoui et al. (2005), provient de la constatation que l'ordre des tâches successives dans une colonne de la solution optimale est, souvent, dicté par un ordre pré-établi déjà connu. Par exemple, un chauffeur d'autobus reste plus longtemps dans son véhicule et traverse plusieurs points de relève successifs sur le trajet de l'autobus. En exploitant cette propriété, et en agrégeant certaines tâches du SPP, le problème devient plus petit et plus facile à résoudre. Cette technique permet de réduire le temps de résolution du problème maître dans un contexte de génération de colonnes.

Règle du *Positive Edge*. Introduite par Raymond et al. (2010b), cette règle définit un nouveau critère de choix de variables à pivoter en base pour la méthode du simplexe primal. Elle identifie, avec un pourcentage d'erreur très faible et en n'effectuant que des opérations de calcul simples sur les coefficients de la matrice originale du problème, des variables donnant lieu à des pivots non dégénérés. Ainsi, les variables identifiées avec cette règle qui présentent un coût réduit négatif permettent de diminuer strictement la valeur de l'objectif du problème. Pour ce faire, le critère du Positive Edge utilise la notion de compatibilité d'IPS. Les colonnes cibles sont des colonnes compatibles ayant un coût réduit négatif. La complexité des calculs pour déterminer la compatibilité des colonnes est la même que celle pour déterminer leurs coûts réduits.

2.4 Travaux parallèles

Nous mentionnons ci-après trois travaux effectués en parallèle avec le nôtre. Ces travaux font face à un souci similaire au nôtre : contrer la dégénérescence et favoriser l'intégralité des solutions recherchées.

Minimum Mean Cycle Canceling. Dans le cadre de résolution de problèmes de flot à coûts minimum, Gauthier et al. (2015b) introduisent une version de l'algorithme *Minimum Mean Cycle Canceling* (MMCC) qui ne présente aucune dégénérescence. Dans chacune de ses itérations, l'algorithme fait diminuer strictement le coût tout en maintenant la réalisabilité primale. Grâce à un paradigme similaire à celui d'IPS, l'algorithme est fortement polynomial.

Simplexe en nombres entiers avec décomposition (ISUD)
Dès le début de nos recherches, nous avons constaté qu'IPS représente un algorithme intéressant vue la ressemblance entre son concept de minimalité avec celui de non décomposabilité chez Balas et Padberg. Une adaptation de cet algorithme a donné naissance à un nouvel algorithme, pour la méthode du simplexe en nombres entiers, que nous avons appelé : Simplexe en nombres entiers avec décomposition (ISUD : Integral Simplex Using Decomposition). Dans sa version de base, ISUD présente une décomposition similaire à celle d'IPS. Son algorithme

est aussi similaire à celui d'IPS avec l'addition de conditions d'intégralité au niveau de RP et CP. Cet algorithme est décrit, plus en détail, dans le chapitre 4.

Dans ISUD, CP est responsable de trouver une direction de descente qui mène à une solution entière. Dans le cadre de sa thèse de doctorat et en parallèle avec nos autres travaux, Samuel Rosat a étudié des possibilités pour augmenter le taux de réussite de CP d'ISUD à trouver une telle direction de descente. Ci-après, on résume deux de ses travaux.

Ajout de coupes à ISUD. Rosat et al. (2014) introduisent une adaptation des méthodes des plans coupant au cas d'ISUD. Ils montrent que des coupes qui peuvent être ajoutées au SPP peuvent être transférées à CP d'ISUD. Ils affirment que de telles coupes existent toujours et ils proposent des algorithmes de séparation pour les coupes primales de cycles impairs et de cliques. Ils spécifient que l'espace de recherche de ces coupes peut être restreint à un petit nombre de variables, ce qui rend leur approche efficace.

Changement de contrainte de normalisation dans ISUD. La contrainte de normalisation (ou contrainte de convexité) est une contrainte à double rôle. Elle garantit que CP soit borné et elle favorise l'intégralité des solutions vers lesquelles mènent les directions que ce problème trouve. Rosat et al. (2016) généralisent cette contrainte dans la formulation de CP et montrent que la nature des directions trouvées par CP et, par conséquent, la probabilité que les solutions vers lesquelles elles mènent soient entières, dépend fortement du choix des coefficients de cette contrainte. Ils en proposent et comparent quelques variantes.

CHAPITRE 3 ORGANISATION DE LA THÈSE

Les trois chapitres de cette thèse qui suivent présentent trois versions d'un algorithme de la famille du simplexe en nombres entiers. Chaque version traite un aspect particulier touchant la performance globale de l'algorithme.

Pallier la dégénérescence. Les efforts effectués pour aboutir à une méthode de simplexe en nombres entiers ont donné naissance à différents algorithmes. Tous ces algorithmes rencontrent la même et principale difficulté : la dégénérescence.

Or, IPS proposé par Elhallaoui et al. (2011) permet d'augmenter considérablement l'efficacité de la méthode du simplexe face à la dégénérescence en utilisant une méthode de décomposition. Le succès d'IPS contre la dégénérescence a donc donné naissance à l'idée suivante : est-ce que la technique de décomposition d'IPS, avec possiblement certaines adaptations, aurait autant de succès envers la dégénérescence rencontrée au niveau des premiers algorithmes du simplexe en nombres entiers ?

L'objectif principal de ce travail est donc de vérifier si cette question a une réponse positive et, dans tel cas, en trouver la meilleure adaptation d'IPS et tester sa performance sur des SPPs de grande taille.

Le chapitre 4 décrit notre algorithme ISUD, pour la méthode du simplexe en nombres entiers, qui est une version adaptée et simplifiée d'IPS. L'algorithme exploite la décomposition d'IPS pour favoriser l'intégralité des solutions au niveau de ses problèmes réduit et complémentaire.

Les premiers résultats, suite à l'implémentation et tests de différentes variantes d'ISUD en nombres entiers, ont vite montré que la décomposition d'ISUD favorise intrinsèquement l'intégralité des solutions produites. Et ISUD, sous sa forme la plus standard, sans aucune adaptation supplémentaire, est déjà un moyen très efficace pour trouver facilement les combinaisons de colonnes recherchées par Balas et les autres pour améliorer une solution entière quand il n'existe plus de variable permettant un pivot améliorant la solution tout en conservant l'intégrité.

Déléguer le branchement. Notre algorithme, dans sa version initiale, donne des résultats intéressants avec un simple branchement en profondeur. Ces résultats peuvent être encore meilleurs si nous pouvons utiliser un branchement sophistiqué dans les cas où les solutions de CP ne sont pas entières. Le chapitre 5 décrit l'amélioration que nous avons introduite pour rendre l'algorithme plus efficace en face d'une situation pareille. En fait, le contrôle d'intégralité, dans la version initiale, est géré par CP. L'amélioration consiste à généraliser la

décomposition héritée d'IPS et proposer un contexte plus général dans lequel la responsabilité de contrôler l'intégralité des solutions est transférée à RP. CP ne s'occupe que de trouver une direction de descente qui indique un voisinage potentiel dans lequel RP cherche une solution améliorée. RP est plus adapté pour le contrôle de l'intégralité dans ce nouveau paradigme de décomposition et peut déléguer la tâche du branchement allégé à un solveur commercial (CPLEX).

Réduire le nombre d'itérations. Dans le chapitre 6, nous cherchons à augmenter la performance de notre algorithme. Pour ce faire, nous observons que cela est possible si nous arrivons à réduire le nombre d'itérations nécessaires pour que l'algorithme atteigne la solution optimale. Nous proposons alors un nouveau modèle pour CP où, au lieu de chercher une direction de descente menant à une solution entière voisine, il cherche directement une solution non nécessairement voisine de meilleur coût. Le nouveau modèle présente plusieurs avantages par rapport au précédent. Les directions retournées par CP ne sont plus forcées à être minimales et l'algorithme peut trouver des solutions subséquentes présentant de bonnes améliorations de coût par rapport aux solutions qui les précèdent. L'algorithme avance donc vers la solution optimale avec de grands pas en moins d'itérations. Ceci donne des résultats impressionnants, dans le cadre de nos tests, lors de la résolution d'instances de grande taille de problèmes de planification d'horaires de chauffeurs d'autobus.

Le chapitre 7 discute les avantages, les limitations ainsi que les possibilités d'extensions de notre algorithme. Le chapitre 8 est une conclusion de cette thèse.

CHAPITRE 4 ARTICLE 1 : INTEGRAL SIMPLEX USING DECOMPOSITION FOR THE SET PARTITIONING PROBLEM

Integral Simplex Using Decomposition for the Set Partitioning Problem

Abdelouahab Zaghrouti, François Soumis, Issmail El Hallaoui

Article publié dans Operations Research (Zaghrouti et al. (2014))

4.1 Introduction

Consider the set partitioning problem (SPP)

$$\begin{array}{ll} \min & cx \\ (\mathbb{P}) & Ax = e \\ & x_j \quad \text{binary}, \quad j \in \{1, \ldots, n\}, \end{array}$$

where A is an $m \cdot n$ binary matrix, c an arbitrary $n-$vector, and $e = (1, \ldots, 1)$ an $m-$vector. Let $(\mathbb{P}')$ be the linear program obtained from $(\mathbb{P})$ by replacing x_j binary with $x_j \geq 0$, $j \in \{1, \ldots, n\}$. We can assume without loss of generality that A is full rank and has no zero rows or columns and does not contain identical columns. We denote by A_j the jth column of A. Also, for a given basis B we denote $\bar{A}_j = (\bar{a}_{ij})_{1 \leq i \leq m} = B^{-1} A_j$.

This article uses some results of Balas & Padberg (1972), which are presented in detail in Section 4.2.1. We first use their results on the existence of a sequence of basic integer solutions, connected by simplex pivots, from any initial integer solution to a given optimal integer solution. The costs of the solutions in this sequence form a descending *staircase*. The vertical section of a step corresponds to a basic change leading to a decrease in the cost of the solution. Each step generally contains several basic solutions associated with the same extreme point of the polyhedron of $\mathbb{P}'$. These solutions are obtained by a sequence of degenerate pivots forming the horizontal section of a step.

We also use the results of Balas & Padberg (1975) on the necessary and sufficient conditions for a set of columns to ensure a transition from one integer solution to another one when they are entered into the basis. A minimal set satisfying these conditions permits to move to an adjacent integer extreme point. The authors also present an enumeration method to find minimal sets that allow the transition from one integer solution to a better adjacent integer solution.

Several other authors have presented enumeration methods that move from one integer solution to a better adjacent integer solution: Haus et al. (2001); Thompson (2002); Saxena (2003), and Rönnberg & Larsson (2009). However, these enumeration methods, moving from a basis to an adjacent (often degenerate) basis, need in practice computation times growing exponentially with the size of the problem, mainly due to severe degeneracy.

This article also uses results on the improved primal simplex (IPS) algorithm for degenerate linear programs (Elhallaoui et al. (2011); Raymond et al. (2010a)). These results are presented in more detail in Section 4.2.2. This algorithm separates the problem into a reduced

problem containing nondegenerate constraints and a complementary problem containing degenerate constraints. The latter generates combinations of variables that improve the reduced problem solution. These variable combinations are minimal because they do not contain strict subsets that permit the improvement of the solution.

Section 4.3 starts with a global view of the Integral Simplex Using Decomposition algorithm (ISUD) we propose in this paper and contains theoretical results proving its validity. Each subsection explains and justifies a procedure of the algorithm. We discuss the relationships between the combinations of variables generated by the complementary problem of IPS and the minimal sets of Balas & Padberg (1975). We present the conditions to be added to the complementary problems to obtain combinations of columns that permit us to move from one integer solution to a better one. Interestingly, the generated combinations often satisfy the conditions and are generally small. When the conditions are not satisfied, we can use a branching method for the complementary problem to satisfy them. This leads to an efficient method for obtaining combinations of columns that permit moving from an integer solution to a better one.

Section 4.4 presents improvements to Balas and Padberg theoretical results on the sequence of adjacent integer bases permitting to move from an initial integer solution to an optimal integer solution. We show that ISUD uses ordinary pivots on positive coefficients only.

Section 4.5 presents ideas permitting to obtain an efficient first implementation of the algorithm. This section contains empirical observations on which are based some strategies to speed up the algorithm.

Section 4.6 presents numerical results for bus driver and aircrew scheduling problems with up to 500 000 variables and 1600 constraints. We show that ISUD is able to find (but not necessarily prove) optimal solutions to the hardest instances in less than 20 minutes. CPLEX was not able to find any feasible solution to such difficult instances in 10 hours.

4.2 Preliminaries

4.2.1 Results from Balas and Padberg

The 1972 paper established the following results. Two bases for a linear program are called adjacent if they differ in exactly one column. Two basic feasible solutions are called adjacent if they are adjacent vertices of the convex polytope that is the feasible set. This distinction is necessary, since two adjacent bases may be associated with the same solution, and two adjacent (basic) feasible solutions may be associated with two nonadjacent bases.

The solution associated with a feasible basis B is integer if and only if there exists Q, a subset of columns of B, such that $\sum_{j \in Q} \quad A_j = e$. If A is of full row rank, every feasible integer solution to $\mathbb{P}'$ is basic.

For $i = 1, 2$, let x^i be a basic feasible integer solution to $\mathbb{P}'$, B_i an associated basis, I_i and J_i the basic and nonbasic index sets, and $Q_i = \{j | x_j = 1\}$.

Theorem 3.1 (Balas & Padberg (1972))
If x^2 is an optimal solution to $\mathbb{P}$, then there exists a sequence of adjacent bases B_{10}, B_{11}, B_{12}, ..., B_{1p} such that $B_{10} = B_1, B_{1p} = B_2$, and

a) the associated basic solutions $x^1 = x^{10}, x^{11}, x^{12}, \ldots, x^{1p} = x^2$ are all feasible and integer,
b) $c \cdot x^{10} \geq \ldots \geq c \cdot x^{1p}$, and
c) $p = |J_1 \cap Q_2|$.

In the proof of this theorem we see that the column pivoted in may have null reduced cost ($\bar{c}_j = 0$) and pivots on negative $\bar{a}_{ij}$ may appear.

The 1975 paper observed the difficulty of identifying the improving sequence of pivots when the optimal solution x^2 is unknown. The authors quoted: "Since set partitioning problems tend to be highly degenerate, the feasible polytope usually contains an enormous number of vertices adjacent to a given vertex. Furthermore, lexicographic or similar techniques are of no avail in coping with degeneracy, since the sequence of pivots required to reach an adjacent vertex may include pivots on a negative entry in a degenerate row."

The article establishes relationships between two adjacent or nonadjacent integer solutions, and the sets of columns on which we must carry out pivots in order to move from one to the other.

Theorems 1 and 2 (Balas & Padberg (1975)) (without the Tucker tableau notation)
Let x^1 be a basic integer solution and I_1, J_1, and Q_1 the associated index sets.

There exists $Q \subseteq J_1$ such that

$$Q^+ = \{k | \sum_{j \in Q} \quad \bar{a}_{kj} = 1\} \quad \subseteq \quad Q_1, \tag{4.1}$$

$$Q^- = \{k | \sum_{j \in Q} \quad \bar{a}_{kj} = -1\} \quad \subseteq \quad I_1 \cap \bar{Q}_1 \tag{4.2}$$

$$\sum_{j \in Q} \quad \bar{a}_{kj} = 0 \quad \subseteq \quad I_1 \setminus \{Q^+ \cup Q^-\} \tag{4.3}$$

if and only if

$$x_j^2 = \begin{cases} 1 & \text{if } j \in Q \cup Q^- \cup (Q_1 - Q^+) \\ 0 & \text{otherwise} \end{cases}$$

is a feasible integer solution obtained from x^1 by performing pivots on Q.

Remark 4.2.1. *The columns associated with (entering) variables in $Q \cup Q^-$ are disjoint, i.e. not covering the same set partitioning constraints and these entering variables replace those in Q^+ (the leaving variables).*

A set $Q \subseteq J_1$ for which (4.1)-(4.2) hold will be called decomposable if Q can be partitioned into subsets Q^* and Q^{**} such that (4.1)-(4.3) hold for both Q^* and Q^{**}. That means that columns with indexes in $Q \cup Q^-$ will replace those with indexes in Q^+ in x^1 in order to obtain an improved solution x^2.

Theorem 3 (Balas & Padberg (1975))
Let x^1 and x^2 be two integer solutions to $\mathbb{P}$, with $Q = J_1 \cap Q_2$. Then x^2 is adjacent to x^1 if and only if Q is not decomposable.

The paper (Balas & Padberg (1975)) describes procedures for generating all-integer vertices of x adjacent to a given vertex. Unfortunately, the procedures are combinatorial and can not solve large problems.

Corollaries 3.2, 3.3, and 3.5 (Balas & Padberg (1975))
Let x^1 and x^2 be two nonadjacent vertices in the feasible domain of $\mathbb{P}$ related to each other by pivots on $Q = J_1 \cap Q_2$.

a) Then there exists a partition of $Q, Q = \cup_{i=1}^{h} Q_i$, such that Q_i satisfies (4.1)-(4.2) and Q_i is not decomposable..
b) For any $H \subseteq \{1, \ldots, h\}$ the pivots on $\cup_{i \in H} Q_i$ starting from x^1 reach a feasible integer solution to $\mathbb{P}$.
c) Any permutation of $\{1, \ldots, h\}$ defines a path of adjacent vertices from x^1 to x^2.

4.2.2 Results from the improved primal simplex (IPS)

We present the results of Elhallaoui et al. (2011) using the notation introduced by Balas & Padberg (1972, 1975), and Raymond et al. (2010a). We discuss these results for $\mathbb{P}'$, the linear relaxation of the set partitioning problem, although they were developed for a general linear programming problem.

The following notation is used **(please note the *I* and *J* used in this section for illustrating IPS are different from the *I* and *J* used by Balas and Padberg)**. If

$x \in R^n$ and $I \subseteq \{1, \ldots, n\}$ is an index set, x_I denotes the subvector of x indexed by I. Similarly for the $m \cdot n$ matrix A, A_I denotes the $m \cdot |I|$ matrix whose columns are indexed by I. If $J = \{1, \ldots, n\} \backslash I, \quad x = (x_I, x_J)$ is used even though the indices in I and J may not appear in order. In the same way the superior indices denote the subset of rows. The vector of all ones with dimension dictated by the context is denoted e.

The paper defines a reduced problem with nondegenerate constraints. This problem is defined according to a feasible basis B that provides the basic feasible solution $\bar{x} = B^{-1}b$. Let P be the index set of the p positive components of this solution. Most of the time, a basic solution to a set partitioning problem is degenerate and $p < m$. Let N be the index set of the zero basic variables. With the partition (P, N) of the constraints and after multiplying by B^{-1} the linear system $Ax = b$ becomes

$$\begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} \begin{bmatrix} x^P \\ x^N \end{bmatrix} = \begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix}, \tag{4.4}$$

with $\bar{b}^N = 0$, $\bar{b}^P > 0$, and $\bar{A}^N_j = 0$, $\forall j \in P$.

Definition 4.2.2. *The j^{th} variable x_j of $\mathbb{P}'$ is said to be compatible with this basis if and only if $\bar{A}^N_j = 0$. Otherwise, it is said to be incompatible.*

It is worthy to outline, as observed in Metrane et al. (2010), that a variable is equivalently said to be compatible with a basis if it is linearly dependent on the positive (nondegenrate) variables (columns) of this basis. Please note that the terms "variable" and its associated "column" are used interchangeably throughout the paper. It is this observation that is used to prove Proposition 6. Let $C \subseteq \{1, \ldots, n\}$ be the index set of compatible variables and $I = \{1, \ldots, n\} \backslash C$ the index set of incompatible variables. x_C and x_I are the subvectors of x of compatible and incompatible variables, respectively. The cost vector is partitioned into c_C and c_I and the columns of A into A_C and A_I.

The linear system (4.4) can be rewritten as follows:

$$\bar{A}^P_C x_C + \bar{A}^P_I x_I = \bar{b}^P$$
$$\bar{A}^N_C x_C + \bar{A}^N_I x_I = 0$$

The reduced problem RP is obtained by imposing $x_I = 0$. Hence,

$$\min_{x_C} \quad c_C \cdot x_C \quad \text{subject to} \quad \bar{A}^P_C x_C = \bar{b}^P, \quad x_C \geq 0 \quad \text{(RP)}.$$

When $x_I = 0$ the constraints indexed by N are satisfied because $\bar{A}^N_C = 0$.

Observations:

- A compatible variable with negative reduced cost enters into the basis with a nondegenerate pivot and a lower cost solution is obtained.
- An optimal solution x^*_C of RP can easily be obtained with the primal simplex. Each pivot moves from an extreme point to an adjacent extreme point with a lower cost. The set P is reduced if degenerate constraints appear.
- Compared to a previous feasible solution, $x = (x^*_C, 0)$ is an improved feasible solution to $\mathbb{P}'$.

In Section 4.3 we will discuss the optimization of RP when the starting solution $\bar{x}$ is integer. The complementary problem is defined when RP is optimal and P, N, C, and I have been readjusted if necessary. The complementary problem is defined with the degenerate constraints, the incompatible variables, and the reduced cost vector $\bar{c}_I$ in the following way:

$$Z^{CP} = \min_v \quad \bar{c}_I \cdot v \quad \text{subject to} \quad \bar{A}^N_I v = 0,\ e \cdot v = 1,\ v \geq 0 \quad (CP)$$

where $\bar{c}_I = c_I - c_P \bar{A}^P_I$.

The one-norm bounding constraint (e$\cdot v = 1$) called also convexity constraint plays an important role in the efficiency of the proposed algorithm, as will be discussed in Section 4.5. It helps finding small combinations of columns that are often disjoint.

Proposition 2 (Elhallaoui et al. (2011))
If CP is infeasible or $Z^{CP} \geq 0$ then $(x^*_C, 0)$ is an optimal solution to $\mathbb{P}'$.

Lemma 1 (Elhallaoui et al. (2011))
If CP is feasible, v^* is an optimal solution with $Z^{CP} < 0$, and S is the subset of indices for which $v^*_j > 0$, then adding the variables in S to RP reduces the cost of the solution.

Proposition 3 (Elhallaoui et al. (2011))
The convex combination $w = \sum_{j \in S} A_j v^*_j$ is compatible with RP and the set S is minimal in the sense that no convex combination of a strict subset of S is compatible with RP.

Elhallaoui et al. (2010) presented an efficient algorithm for the linear relaxation of the set partitioning problem using compatible variables and a reduced problem RP. This paper contains an interesting result for us.

Proposition 5.6 (Elhallaoui et al. (2010))
If x^1 is a nondegenerate feasible integer solution to RP and if there exists a compatible variable with a negative reduced cost, then the new solution obtained by pivoting on this

variable is an improved integer solution.

4.3 The integral simplex using decomposition algorithm

Balas & Padberg (1972) showed that between two integer solutions there exists a sequence of integer solutions with nonincreasing costs for which the associated bases are adjacent. However, some pivots between adjacent bases must be carried out on negative coefficients. We propose the Integral Simplex Using Decomposition (ISUD), a new algorithm based on the IPS decomposition to find a sequence of adjacent basic solutions of non increasing costs leading to optimality. Furthermore, the algorithm carries out simplex pivots only on the positive coefficients. This point is discussed in Section 4.4. Here is a global view of the ISUD algorithm:

Step 0: Start with an initial integer solution.

Step 1:

- — Improve the current integer solution with efficient pivots in the reduced problem (see Subsection 4.3.1 for details).
- — Each pivot produces a better integer solution by increasing one variable. This creates a step in the staircase with a single basis change to move to a better adjacent extreme point.
- — When there is no improvement with Step 1, go to Step 2.

Step 2:

- — Improve the current integer solution with a solution to the complementary problem: a group of disjoint variables having (together) a negative reduced cost (see Subsection 4.3.2 for details).
- — This step produces a better integer solution by increasing many variables. This creates a step in the staircase with many basis changes by degenerate pivots before a nondegenerate pivot, permitting to move to a better adjacent extreme point, is executed.
- — Some branching can be necessary in this step to obtain a column-disjoint solution to CP (see Subsection 4.3.3 for details).

Control step:

- — If Step 2 improves the solution, go to Step 1.
- — If there is no more improvement with Step 2, the integer solution is optimal (see Subsection 4.3.4 for details).

We first add a result to the work of Balas & Padberg (1972). The result is an extension of a basic result of linear programming on the transition between two adjacent basic solutions.

When x^2 is obtained from x^1 by a simplex pivot entering x_j into the basis at value 1, we know that

$$c * x^2 = c * x^1 + \bar{c}_j = c * x^1 + c_j - c_B \bar{A}_j$$

where c_B is the cost of the basic variables and $\bar{c}_j$ is the reduced cost of variable j. We extend this result to a move from x^1 to x^2 with a sequence of pivots on variables of a set Q. Generally, some reduced costs are modified after each pivot and the cumulative effect can be complex. In our case, the properties of the set Q as explained below permit to prove the following result.

Proposition 4.3.1. *If x^1 and x^2 are two integer solutions related by a sequence of pivots on the variables of a set Q satisfying (4.1)-(4.3), then*

$$c \cdot x^2 = c \cdot x^1 + \sum_{j \in Q} c_j - c_B \cdot \sum_{j \in Q} \bar{A}_j$$

Proof. Consider the aggregated variable that is the sum of the (entering) variables in $Q \cup Q^-$. Its reduced cost is $\sum_{j \in Q \cup Q^-} \bar{c}_j$. As outlined in Remark 4.2.1, pivoting on variables in Q has the same effect as pivoting on the aggregated variable. So, we have

$$c * x^2 = c * x^1 + \sum_{j \in Q \cup Q^-} \bar{c}_j$$

As the reduced cost of variables in Q^- is null because they are basic, we obtain the desired result:

$$c \cdot x^2 = c \cdot x^1 + \sum_{j \in Q} c_j - c_B \cdot \sum_{j \in Q} \bar{A}_j$$

□

4.3.1 Improvement of the current integer solution by RP

This subsection explains and justifies Step 1 of the algorithm. The IPS method is here specialized for set partitioning problems starting with an initial integer solution. In this case, we will see that the optimization of the reduced problem involves carrying out pivots on variables that move from one integer solution to a better one.

Let x be an integer solution to the set partitioning problem and P the index set of its positive components. Define an associated basis for which the first p variables are those of P and the

remaining $(m-p)$ are artificial variables of cost M. M is sufficiently large to ensure that the artificial variables do not appear in an optimal solution. The constraints are also partitioned into (P, N) where P is the index set of the nondegenerate constraints and N the index set of the degenerate constraints. The same set P is used for variables and constraints because it is the same set after reordering the constraints and variables to obtain $A_P^P = I$. The basis with this partition is

$$B = \begin{bmatrix} A_P^P & A_N^P \\ A_P^N & A_N^N \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_P^N & I \end{bmatrix}.$$

$$B^{-1} = \begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \quad \text{because}$$

$$\begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \quad \begin{bmatrix} I & 0 \\ A_P^N & I \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_P^N - A_P^N & I \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}.$$

Multiplying by B^{-1} the system $Ax = b$ becomes

$$\begin{bmatrix} \bar{A}^P \\ \bar{A}^N \end{bmatrix} \quad \begin{bmatrix} x^P \\ x^N \end{bmatrix} = \begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix},$$

where

$$\begin{bmatrix} \bar{A}_j^P \\ \bar{A}_j^N \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \quad \begin{bmatrix} A_j^P \\ A_j^N \end{bmatrix} = \begin{bmatrix} A_j^P \\ -A_P^N & A_j^P + A_j^N \end{bmatrix}, \tag{4.5}$$

$\bar{A}_j^N = 0 \quad \forall j \in P$ because $A_P^P = I$, and

$$\begin{bmatrix} \bar{b}^P \\ \bar{b}^N \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A_P^N & I \end{bmatrix} \quad \begin{bmatrix} b^P \\ b^N \end{bmatrix} = \begin{bmatrix} b^P \\ -A_P^N & b^P + b^N \end{bmatrix} = \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

By definition, the j^{th} variable x_j of $\mathbb{P}$ is said to be compatible with the basis B if and only if $\bar{A}_j^N = 0$.

Using C and I, the index sets of compatible and incompatible variables, x_C and x_I are a partition of the variables, c_C and c_I are a partition of the variable costs, and A_C and A_I are a partition of the columns of A. The reduced problem RP is obtained by imposing $x_I = 0$. It is:

$$\min_{x_C} \quad c_C \cdot x_C \quad \text{s.t.} \quad \bar{A}_C^P x_C = \bar{b}^P,\ x_C \geq 0. \quad (RP)$$

In this case the basis $A_P^P = I$, $\bar{b}^P = e$, and $x_P = I x_P = e$.

Proposition 4.3.2. *If x is a nonoptimal integer solution to RP then there exists a compatible*

variable with negative reduced cost and the new solution obtained by pivoting on this variable is an improved integer solution.

Proof. If x is nonoptimal then there exists a compatible variable with negative reduced cost. Furthermore, if x is integer, feasible, and nondegenerate then by Proposition 5.6 (Elhallaoui et al. (2010)) the new solution obtained by pivoting on this variable is an improved integer solution. □

Proposition 4.3.3. *If we start with a feasible integer solution to RP, x^1, and apply the simplex algorithm while eliminating the degenerate constraints and the resulting incompatible columns as they appear, we generate a sequence $x^1, x^2, \ldots, x^k$ of solutions with the following properties*

1. $c \cdot x^1 > c \cdot x^2 > , \ldots, > c \cdot x^k$,
2. $x^1, x^2, \ldots, x^k$ *are integer solutions of* $\mathbb{P}$,
3. x^k *is an optimal solution to the linear relaxation of the final RP problem.*

Proof. Given x^i, let RP^i be the associated reduced problem. If there exists a compatible variable with a negative reduced cost, the solution x^{i+1} after a simplex pivot is integer and $cx^i > cx^{i+1}$ by Proposition 4.3.2. RP^{i+1} is defined by eliminating the degenerate constraints if they exist. The process continues while there is at least one compatible variable with a negative reduced cost.

The solutions x^i of the RP^i problems are solutions of $\mathbb{P}$ if we complete them by adding incompatible variables set to zero values.

x^k is an optimal solution to the linear relaxation of the problem RP^k because there are no more compatible variables with negative reduced costs. □

However, this optimal solution to RP might not be an optimal solution to $\mathbb{P}$. In the following subsection, we discuss how to improve it.

4.3.2 Improvement of the current integer solution by CP

This section explains and justifies Step 2 of the algorithm in the basic case when the branching is not necessary. Section 4.3.3 discusses the branching procedure called when necessary.

We now suppose that x is an optimal integer solution to RP and RP has been reduced as necessary so that x is nondegenerate, and B is the associated basis after completing P with artificial variables. The optimization of the complementary problem is used to find variables

that permit the improvement of the integer solution if they are all entered into the current basis. The complementary problem CP is

$$Z^{CP} = \quad \min \quad \bar{c}_I \cdot x_I \quad \text{s.t.} \quad \bar{A}_I^N x_I = 0,\, e \cdot x_I = 1,\, x_I \geq 0. \quad (CP)$$

In this case,

$$\begin{aligned} \bar{A}_I^N &= -A_P^N A_I^P + A_I^N, \\ \bar{c}_I &= c_I - \left(A_P^P\right)^{-1} A_I^P c_P = c_I - A_I^P c_P. \end{aligned}$$

Proposition 4.3.4. *If x_C^* is an optimal integer solution to RP and CP is infeasible or $Z^{CP} \geq 0$ then $(x_C^*, 0)$ is an optimal solution to $\mathbb{P}$.*

Proof. $(x_C^*, 0)$ is an optimal solution to $\mathbb{P}'$ by Proposition 4.3.2 (Elhallaoui et al. (2011)). Since $\mathbb{P}'$ is a relaxation of $\mathbb{P}$, this integer solution is an optimal solution to $\mathbb{P}$. □

Let x_I^* be an optimal solution to CP and S the set of indices $j \in I$ such that $x_j^* > 0$.

Proposition 4.3.5. *The convex combination $w = \sum_{j \in S} A_j x_I^*$ is compatible with RP, and the set S is minimal in the sense that no convex combination of a strict subset of S is compatible with RP.*

Proof. This follows from Proposition 4.3.3 (Elhallaoui et al. (2011)). □

We introduce the following definition before establishing the relationship between the sets S of IPS and the sets Q of Balas and Padberg.

Definition 4.3.6. *Let $S \subseteq I$ be a set of columns, then S is column-disjoint if $A_{j_1} \cdot A_{j_2} = 0, \quad \forall (j_1, j_2) \in S \times S, j_1 \neq j_2$.*

For Propositions 4.3.7, 4.3.8, EC.1, and EC.2, consider the following context: Let x_C^* be an optimal solution to RP having a set of constraints P, x_I^* an optimal solution to CP, and S the set of indices $j \in I$ such that $x_j^* > 0$. Consider the case where $\bar{c}_I . x_I^* < 0$ and S is column-disjoint.

Proposition 4.3.7. *The value the nonzero variables $x_j^*, j \in S$ take in an optimal solution to CP is $x_j^* = \frac{1}{|S|}$ for $j \in S$.*

Proof. Every solution to CP is compatible. So, we can say that this solution is a linear combination of the positive variables of the basis, i.e. the columns of S. Mathematically speaking, $\sum_{j\in S} x_j A_j = \sum_{k\in P} \lambda_k A_k$, where $(\lambda_k)_{k\in P}$ are reals. The normalization constraint must be satisfied, meaning that $\sum_{j\in S} x_j = 1$. The optimal solution to CP we are looking for is integer, i.e. its columns are disjoint. We also know that the current solution to RP (and also to the original problem) is integer. So, we can say that the columns of this solution are disjoint too. From all of that, we easily can see that $x_j^* = \lambda_k^* = \frac{1}{|S|}, j \in S, k \in P$ is a feasible solution to CP.

We know, as highlighted in Section 4.2.2, that every solution to CP is minimal. Using the minimalistic property, and as proven in Elhallaoui et al. (2011), we can confirm that this solution to CP is unique, in a sense that $x_j^*, j \in S$ must take $\frac{1}{|S|}$ as value. □

Proposition 4.3.8. *S corresponds to a nondecomposable set Q (Balas & Padberg (1975)) that permits us to obtain x^{**}, a new lower-cost integer solution to $\mathbb{P}$ and $Q^- = \emptyset$.*

Proof. We consider $\sum_{j\in S} \bar{A}_j^k$ to determine the variables that are modified in order to obtain a feasible solution when we set to 1 the variables with indices in S.

For $k \in N$

$$\sum_{j\in S} \bar{a}_{kj} = |S| \sum_{j\in S} \bar{a}_{kj} x_j^* = 0.$$

For $k \in P$, we have $\bar{a}_{kj} = a_{kj}$ by 4.5 and $\sum_{j\in S} a_{kj}$ is binary because the columns $A_j, j \in S$ are disjoint.

Let

$$S^+ = \{k | \sum_{j\in S} \bar{a}_{kj} = 1\} \subseteq P,\ Q^- = S^- = \{k | \sum_{j\in S} \bar{a}_{kj} = -1\} = \emptyset.$$

S satisfies the condition (4.1)-(4.2) of Theorems 1 and 2 of Balas & Padberg (1975) and

$$x_j^{**} = \begin{cases} 1, & j \in S \cup S^- \cup (P - S^+) = S \cup (P - S^+) \\ 0, & \text{otherwise} \end{cases}$$

is a feasible integer solution to $\mathbb{P}$.

We now calculate the cost of solution x^{**}. By Proposition 4.3.1, we have

$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j \in S} c_j - c_B \sum_{j \in S} \bar{A}_j,$$
$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j \in S} c_j - c_P \sum_{j \in S} A_j^P,$$

because

$$\{k | \sum_{j \in S} \bar{a}_{kj} \neq 0\} = S^+ \subseteq P \subseteq B,\ \bar{A}_j^N = 0,\ j \in S \quad \text{and} \quad \bar{A}_j^P = A_j^P,\ j \in S.$$
$$c \cdot x^{**} = c \cdot (x_c^*, 0) + \sum_{j \in S} \bar{c}_j \quad \text{because} \quad \bar{c}_j = c_j - c_P \quad A_j^P$$
$$c \cdot x^{**} = c \cdot (x_c^*, 0) + |S| \cdot \bar{c}_I x_I^*$$

Therefore,

$$c \cdot x^{**} < c \cdot (x_c^*, 0) \quad \text{because} \quad \bar{c}_I \cdot x_I^* < 0.$$

Thus, we obtain a new lower-cost integer solution. □

Remark 4.3.9. *The sets S generated by the CP are a particularly interesting case of the sets Q of Balas & Padberg (1975).*

Since $S^- = \emptyset$, we move from the integer solution $(x_c^*, 0)$ to the adjacent integer solution x^{**} by adding only the set S to the variables equal to 1. We do not set to 1 basic variables that were previously 0. The variables of S^+ change from 1 to 0 as in Balas & Padberg (1975).

4.3.3 Branching to obtain column-disjoint solutions to CP

When the solution to CP is not column-disjoint, we use the following branching method. Let V be the columns of S that have a nonempty intersection with another column of S; $V = \{j | j \in S$, there exists $i \in S$ such that $A_i \cdot A_j \neq 0\}$. $V \neq \emptyset$ because S is not column-disjoint. We define $|V| + 1$ branches, a branch 0 and a branch j for each $j \in V$:

Branch 0: $x_j = 0$ for $j \in S$.

These constraints are easy to impose. It is sufficient to remove the variables set to zero from CP.

Branch j: $x_i = 0$ for $i \in I$, $i \neq j$ and $A_i \cdot A_j \neq 0$ and x_j free.

These constraints are easy to impose. It is sufficient to remove variables set to zero

from CP. It would be good to impose $x_j \neq 0$ in order to have disjoint branches but this is not easy. We can not impose $x_j = 1$ because in any solution to CP we have $x_j < 1$. We can not impose $x_j = \frac{1}{|S|}$ because the size of S can change at the next solution of CP.

All these branches eliminate the current solution $x_j > 0$, $j \in S$ because we fix to zero at least one variable of S. When exploring the branches, the algorithm gives priority to the zero branch. The idea is that solutions to CP have a high probability of being column-disjoint. If a solution is not, we look elsewhere where it probably is. If the solution to CP in branch 0 is not column-disjoint we branch again and explore the next zero branch. This deep search continues as long as the marginal cost of the solution of CP is negative. If we must explore the j branches to obtain a solution, we start with j having minimum $\bar{c}_j$.

It is not necessary to explore the full branching tree. As soon as a column-disjoint solution with a negative marginal cost is obtained in some branch, it can be added to RP to improve the current integer solution. The current tree is abandoned and a new tree is considered at the subsequent iteration.

4.3.4 Improving the current integer solution by CP is sufficient to reach optimality

This section justifies the control step of the algorithm. It is sufficient to prove that Step 2 permits to reach optimality. A pivot in Step 1 is a special case of Step 2 where the set Q reduces to a single column. Step 1 is a faster way to improve the integer solution when it is possible.

The following proposition shows that the sets Q with $Q^- = \emptyset$ are sufficient to move from an integer solution to $\mathbb{P}$ to an optimal solution to $\mathbb{P}$. For $i = 1, 2$, let x^i be an integer solution to $\mathbb{P}$, I_i and J_i the basic and nonbasic index sets, and $Q_i = \{j | x^i_j = 1\}$.

Proposition 4.3.10. *If x^1 is a nonoptimal integer solution to $\mathbb{P}$ and x^2 is a lower-cost integer solution to $\mathbb{P}$ then there exists $Q \subseteq J_1$ such that*

1. $Q^+ = \{k | \sum_{j \in Q} \bar{A}^k_j = 1\} \subseteq Q_1$,
2. $Q^- = \{k | \sum_{j \in Q} \bar{A}^k_j = -1\} = \emptyset$,
3. $Q_2 = Q \cup Q^- \cup (Q_1 - Q^+)$,
4. *Q is column-disjoint.*

Proof. Let $Q = Q_2 - (Q_1 \cap Q_2)$. To study $Q^+ = \{k | \sum_{j \in Q_2 - (Q_1 \cap Q_2)} \quad \bar{A}_j = 1\}$, we first prove the following lemma. □

Lemma 4.3.11.

$$\sum_{j \in Q_1 - (Q_1 \cap Q_2)} \bar{A}_j = \sum_{j \in Q} \bar{A}_j$$

Proof. First $\sum_{j \in Q_1} A_j = \sum_{j \in Q_2} A_j = e$ because x^1 and x^2 are solutions to $\mathbb{P}$. We obtain by subtracting the A_j, $j \in Q_1 \cap Q_2$:

$$\sum_{j \in Q_1 - (Q_1 \cap Q_2)} A_j \quad = \quad \sum_{j \in Q_2 - (Q_1 \cap Q_2)} A_j \quad = \quad \sum_{j \in Q} A_j.$$

Multiplying by B^{-1}, the inverse of the basis B associated with x^1, we obtain

$$\sum_{j \in Q_1 - (Q_1 \cap Q_2)} \bar{A}_j \quad = \quad \sum_{j \in Q} \bar{A}_j.$$

□

Proof of 1: We have

$$Q^+ = \{k | \sum_{j \in Q_1 - (Q_1 \cap Q_2)} \bar{A}_j = 1\} = Q_1 - (Q_1 \cap Q_2) \subseteq Q_1$$

because

$$\bar{A}_j = e_j \quad \text{for} \quad j \in Q_1, \text{where } e_j \text{ is a column of } I.$$

Proof of 2: We have

$$Q^- = \{k | \sum_{j \in Q} \bar{A}^k_j = -1\} = \{k | \sum_{j \in Q_1 - (Q_1 \cap Q_2)} \bar{A}_j = -1\} = \emptyset$$

because

$$\bar{A}_j = e_j \quad \text{for} \quad j \in Q_1.$$

Proof of 3: We have $Q \cup Q^- \cup (Q_1 - Q^+)$

$$= \Big(Q_2 - (Q_1 \cap Q_2)\Big) \cup \emptyset \cup (Q_1 - (Q_1 - (Q_1 \cap Q_2)))$$

using the definition of Q and the values of Q^- and Q^+.

$$= \Big(Q_2 - (Q_1 \cap Q_2)\Big) \cup (Q_1 \cap Q_2) = Q_2.$$

Proof of 4: Q is column-disjoint because $Q \subseteq Q_2$, which is a basic integer solution of the set partitioning problem.

Let us show that Q of Proposition 4.3.10 defines a column-disjoint solution to CP with a negative reduced cost.

Proposition 4.3.12. *If x^1 is a nonoptimal integer solution to $\mathbb{P}$, x^2 a lower-cost integer solution to $\mathbb{P}$, and Q the set defined in Proposition 4.3.10, then $\widehat{x}_j = \frac{1}{|Q|}, j \in Q$ and $\widehat{x}_j = 0, j \notin Q$ is a feasible solution, with a negative reduced cost, to the complementary problem defined from x^1.*

Proof. For the solution x^1, Q_1 is the index set of the nondegenerate constraints and its complement N is the index set of the degenerate constraints.

$$\sum_{j \in Q} \bar{A}_j = \sum_{j \in Q_1 - (Q_1 \cap Q_2)} \bar{A}_j \qquad \text{by Lemma 4.3.11.}$$

$$= \sum_{j \in Q_1 - (Q_1 \cap Q_2)} e_j \qquad \text{because} \quad \bar{A}_j = e_j, \quad j \in Q_1.$$

Therefore,

$$\sum_{j \in Q} \bar{A}^N_j = \sum_{j \in Q_1 - (Q_1 \cap Q_2)} 0^N = 0 \qquad \text{because} \quad N \cap Q_1 = \emptyset.$$

Hence,

$$\sum_{j \in I} \bar{A}^N_j \widehat{x}_j = \sum_{j \in Q} \bar{A}^N_j \widehat{x}_j = \frac{1}{|Q|} \sum_{j \in Q} \bar{A}^N_j = 0.$$

The last constraint is also satisfied:

$$e \cdot \widehat{x} = \sum_{j \in I} \widehat{x}_j = \sum_{j \in Q} \widehat{x}_j = |Q| \cdot \frac{1}{|Q|} = 1$$

The reduced cost of this solution is $\sum_{j\in Q} \bar{c}_j \cdot \widehat{x}_j = \sum_{j\in Q} \bar{c}_j \frac{1}{|Q|} = \frac{1}{|Q|} \sum_{j\in Q} \bar{c}_j$.

This value is calculated from the result of Proposition 4.3.1:

$$c \cdot x^2 = c \cdot x^1 + \sum_{j\in Q} c_j - c \cdot \sum_{j\in Q} \bar{A}_j$$

Hence,

$0 > cx^2 - cx^1 = \sum_{j\in Q} c_j - c_B \cdot \sum_{j\in Q} \bar{A}_j,$

$0 > \sum_{j\in Q} c_j - c_P \sum_{j\in Q} \bar{A}_j^P - c_N \cdot \sum_{j\in Q} \bar{A}_j^N$ using the row partition (P, N), on the column.

In the basis, a column corresponds to each row.

$0 > \sum_{j\in Q} c_j - c_P \sum_{j\in Q} A_j^P$ because $\sum_{j\in Q} \bar{A}_j^N = 0$ and $\bar{A}_j^P = A_j^P$,

$0 > \sum_{j\in Q} \bar{c}_j.$

The reduced cost of this solution is thus negative. □

Remark 4.3.13. *The set Q from Propositions 4.3.10 and 4.3.12 can be decomposable. In this case, there are nondecomposable subsets Q_i of Q. Proposition 4.3.12 also applies to each set Q_i. Each Q_i defines a minimal feasible solution to the complementary problem.*

4.4 Improvement of Balas and Padberg results on the sequence of adjacent bases leading to optimality

Balas and Padberg proved the existence of a sequence of adjacent bases of non-increasing cost from an initial integer solution to a given better integer solution. Their sequence uses some pivots on negative $\bar{A}_j^k$. ISUD is a constructive approach producing a sequence without pivots on negative $\bar{A}_j^k$ and does not need to know a better a priori integer solution.

4.4.1 The type of pivots used by the ISUD algorithm

As discussed in Section 4.3, the RP can be solved with the simplex algorithm using regular pivots on positive $\bar{A}_j^k$. The pivots on negative $\bar{A}_j^k$ are not necessary with ISUD. In the other algorithms presented in the literature (Balas & Padberg (1972, 1975); Haus et al. (2001); Rönnberg & Larsson (2009); Thompson (2002)), the basis contains zero variables. Degenerate pivots on negative $\bar{A}_j^k$ are used to exchange some zero variables in the basis with

non basic variables. In ISUD, there is no zero basic variables and this kind of pivots is not necessary.

The CP can also be solved with ordinary simplex pivots. It is an ordinary linear programming problem embedded in a branching tree. Because ISUD uses only ordinary simplex pivots the algorithm can be implemented with a regular simplex algorithm program without any internal modification. We implemented it with CPLEX taking advantage of all the knowhow embedded in this black box.

4.4.2 The number of pivots in ISUD

Balas & Padberg (1972, 1975) present a bound on the number of primal pivots in their sequences of adjacent bases between two integer solutions but the proof supposes the knowledge of the final solution. ISUD does not use this information and does not have the same strong result. It is possible to prove that ISUD does exacltly $|S|$ pivots when adding a set S to improve the solution of RP (see Propositions EC.1 and EC.2 in the e-companion). This proof is not for a practical use. Actually, ISUD uses a better and simple solution method to update the current integer solution (see Section 4.5).

When we solve CP with the primal simplex without knowing the set S in advance, more than $|S|$ pivots may be necessary to reach the optimal solution when the entry criterion is to choose the variable with the minimum reduced cost. For example, if there are several sets S_l defining feasible solutions of CP, we can enter into the basis variables from several S_l via degenerate pivots before to reach an optimal solution. Indeed, even if S_{l^*} has a minimal mean marginal cost (i.e. $\sum_{j\in S_{l^*}} \bar{c}_j x_j$ is minimal s.t. $\sum_{j\in S_{l^*}} x_j = 1$) this does not imply that the $\bar{c}_j$ of the set S_{l^*} are inferior to all the $\bar{c}_j$ of the set $I - S_{l^*}$.

The e-companion translates the sequence of basic solutions generated by ISUD into the original polyhedron and discusses the difference with the sequence of Balas and Padberg.

4.5 Implementation tips

This section presents some efficient methods to solve the reduced problem and the complementary problem. For example, the multi-phase strategy explained below permits a significant speed-up of the algorithm. The justifications in this section are based on intuitive discussions, observations and numerical results presented in Section 4.6. Nothing in this section can infirm the validity of the algorithm. The effects are only on its speed.

The RP is a nondegenerate linear program. It can easily be solved with the primal simplex

algorithm. Each pivot improves the solution. Furthermore, the pivots are regular pivots on positive $\bar{A}^k_j$. The implementation of ISUD performs better than the simplex algorithm in solving the RP by using the following remark.

Remark 4.5.1. *In Step 1 and Step 2, modifications to the RP solution can be conducted without using simplex pivots.*

Actually, updating the RP solution (that is also solution to the original problem) either by entering into the basis a single compatible variable x_j (in Step 1), (say $S = \{j\}$) or by many incompatible variables x_j, $j \in S$ (in Step 2), where S is associated with a solution to CP, is done as follows:

— the new solution is obtained by changing from 0 to 1, x_j, $j \in S$ and by changing from 1 to 0 the variable(s) covering the same constraints,
— the new basis is still I,
— the new dual variables are $\pi = c_B B^{-1} = c_B I = c_B$.

And because CP is severely degenerate primal simplex is not a good algorithm to solve it, the dual simplex is a more appropriate algorithm. The solution that sets to 1 the variable with the most negative reduced cost is a good starting dual feasible solution. Another method that is effective for this problem is the Presolve algorithm of CPLEX. It often reduces the CP problem by more than 90% and the solution to the residual problem with the dual is rapid. In Section 4.6, we present numerical experiments on the solution of the complementary problem.

The numerical results show that the set S associated with the optimal solution to CP is not only minimal but generally small. Indeed, for the solutions in the form identified in Proposition 4.3.7, the objective value is $\frac{1}{|S|} \quad \sum_{j \in S} \quad \bar{c}_j$. It is therefore equal to the average cost of the $\bar{c}_j$ in S. This cost is minimal if S contains the smallest $\bar{c}_j$ allowing the constraints to be satisfied. Increasing the set S leads to the addition of larger $\bar{c}_j$ values. We discuss the potential impact of the test problems properties on the size of sets S in the numerical results section (Section 4.6). Another formulation of CP has been considered; it replaces the constraint $\sum_{i \in I} x_i = 1$ by $x_i \leq 1, i \in I$. This formulation produces much larger sets S and do not have the same good properties: S is larger and not minimal, the probability of being column-disjoint is small, and the values of the positive variables can be different of $1/|S|$.

Small S sets are more likely to be column-disjoint. Indeed, we will see in the numerical results that the optimal solutions of CP are often column-disjoint even if this constraint is relaxed. The solution of CP is therefore an effective way to identify sets of columns that permit us to move from one integer solution to an adjacent integer solution of a lower cost.

The multi-phase strategy introduced by Elhallaoui et al. (2010) could be used here to accelerate solving CP. This strategy proceeds through a sequence of phases. Each phase corresponds to a different level of partial pricing. The level is defined by the number of incompatibilities that a column $j \in I$ can have with respect to the partition defined by the integer solution to RP. Let the cluster of this partition defined by the columns for which $x_l = 1$, $l = 1, \ldots, p$ be $W_l = \{k | A^k_l = 1\}$, $l = 1, \ldots, p$. This number can be mathematically defined as follows:

Definition 4.5.2. *Given a partition $W_l = \{k | A^k_l = 1\}$, $l = 1, \ldots, p$ and a column j, the number of incompatibilities of j with respect to W is given by $k^W_j = \sum_{l=1}^{p} k_{lj}$, where k_{lj} is equal to 1 if column j covers some, but not all, of the elements in W_l, and 0 otherwise. A column j and its associated variable x_j are said to be* **k-incompatible** *with respect to W if $k^W_j = k$. Compatible columns and variables are also qualified as 0-incompatible variables.*

A phase of the ISUD algorithm is defined as follows.

Definition 4.5.3. *The ISUD algorithm is said to be* **in phase k** *when, among the variables x_j, only those that are q-incompatible with $q \leq k$ are priced out by CP. k is called* **the phase number**.

The sequence of phases that the algorithm goes through is predetermined and, to ensure the exactness of the algorithm, it must terminate with a phase where all variables are priced out. Observe that p is an upper bound on k^W_j. The advantages of the multi-phase algorithm on the mono-phase algorithm are presented in the experimentation section.

The results in Section 4.6 are produced with the implementation tips described above.

4.6 Experimentation

We tested the ISUD (Integral Simplex Using Decomposition) algorithm on a 2.8 Ghz dual core Linux machine where a single processor is used. The solver used is CPLEX (version 12.0). In our experiments, we run both CPLEX and ISUD on different instances and compared their performances. On the ISUD side, we included results from its mono-phase and multi-phase versions. The mono-phase version refers to the version of ISUD where we do not use the multi-phase strategy.

This first experimentation presents results from a heuristic branching for CP. The branching is a deep search without backtracking; exploring only the zero branch. The exploration stops when the reduced cost of the solution becomes greater or equal to zero. This partial

exploration produces optimal solutions in many cases and near optimal solutions in the majority of the other cases.

4.6.1 Instances

To test the performance of ISUD on problems of different size, we used 3 problems:

— sppaa01 (small): aircrew scheduling problem from OR-Lib ($\sim$ 800 constraints x 8000 variables)

— vcs1200 (medium): randomly generated bus and driver scheduling problem ($\sim$ 1200 constraints x 130000 variables)

— vcs1600 (large): randomly generated bus and driver scheduling problem ($\sim$ 1600 constraints x 500000 variables)

The number of nonzeros per column is in average 9 in sppaa01 and 40 in vcs1200 and vcs1600 instances. These numbers of nonzeros are typical for aircrew and bus driver scheduling problems. They do not increase with the number of constraints. They are related to the number of flights per pairing for aircrews and the number of pieces of work per duty for bus drivers. Thus, these problems become fairly sparse when the number of constraints increases. This should help obtaining disjoint columns in CP solutions more frequently. The ISUD is well adapted to real life large scale vehicle and crew scheduling problems and probably would work better on larger problems.

ISUD needs an initial solution to start from. So, those problems are first solved using CPLEX for the small problem and GENCOL for the medium and large problems. GENCOL is a commercial software developed at the GERAD research center in Montreal and now owned by AD OPT Technologies, a division of KRONOS. GENCOL supports the column generation method and is used widely to solve vehicle and crew scheduling problems. For the set partitioning problem, a greedy heuristic produces infeasible solutions most of the time. We discuss at the end of Section 4.6.2 the results obtained when starting up the ISUD with such greedy poor solutions.

We developed a method to randomly perturb the optimal solution to obtain a feasible initial solution with a certain level of good primal information similar to initial solutions generally available in practice for crew scheduling problems. The columns of the perturbed solution are added to the problem with a high cost. This method permits to obtain many instances (many starting solutions) from an initial problem.

The motivation behind this perturbation method comes from two observations. Firstly, in crew scheduling problems, we observe that crews do not change their vehicles very often; their rotations have thus many parts in common with the vehicle routes. Secondly, for

reoptimization of crew or vehicle schedules during their operation we notice that reoptimized solutions deviate slightly from planned ones. Consequently, many consecutive tasks (flights, bus trips) on the initial *paths* (the vehicle or planned solution paths in aircrew or bus and crew scheduling problems described above) will remain grouped in an optimal solution. The variables are associated with paths (pilot or driver schedules) that are feasible with regards to a set of predefined rules. A path contains an ordered sequence of tasks and possibly other activities (breaks, rests, briefings, ...).

We can consider as a measure of good primal information the percentage of consecutive pairs of tasks in the initial solution that remain grouped in the optimal solution. An initial solution following the bus route as much as possible has generally 90% or more of good primal information. In fact, in an optimal solution a bus driver stays on the same bus over 90% of the relief points, i.e. locations where driver exchanges can occur. Similarly, an initial crew pairing solution following the aircraft route has a high percentage of good primal information. For the short and medium haul problems, crews operate approximately 5 flights a day. Generally, more than 50% of them stay on the same aircraft all the day. The others change the aircraft one or two times the day. Less than 5% changes the aircraft two times. At the end of their working day, 20% operate the same aircraft the following day. For a typical day of 5 flights, we have an average of 1.35 aircraft changes ($.8 \times 1$ at the end of the day, and $.5 \times 0 + .45 \times 1 + .05 \times 2$ during the day). From that, we can conclude that an aircrew changes the aircraft (after a flight) 27% (1.35 / 5) of the time or less. Meaning that an initial solution following the aircraft routes has more that 73% of good primal information.

Below, we present a perturbation process of the optimal solution to create an initial solution with a high percentage of good primal information. The perturbation process of a solution consists in randomly selecting two of its columns and replacing them with two columns covering same tasks as the two original columns. The perturbation process chooses two tasks belonging to two different paths in the solution (set of paths) and connecting them to create a new path.

This process is repeated until the number of unchanged columns (belonging to an initial solution) goes below a certain percentage of the number of columns in the solution. For our tests, we set this number to 50%, 35% and 20%. Also, the newly generated columns are added to the problem and given the maximum cost of all columns in the problem. The perturbed columns are often selected again for supplementary perturbations and this process can go far from the initial optimal solution. Note that the optimal solution columns are not removed from the problem. The percentage of good primal information in the initial solutions we start from is discussed in the following subsection.

4.6.2 Results

Table 4.1 shows the mono-phase version of ISUD and CPLEX results from a set of 10 executions on different instances of the small problem (perturbed to 50%). We read from left to right: the percentage of the optimal solution columns left in the initial solution, the ISUD execution time, the CPLEX execution time, the time ratio comparing the performance of the two algorithms, the initial ISUD error on the optimal objective value (a percentage expressing how far is the optimal value from the one ISUD starts from), the final ISUD error, the number of CPs solved by ISUD, the number of the ones that had disjoint solutions among them, the maximum size of the disjoint solutions and their average. The bottom line contains column averages.

The mono-phase version was able to find disjoint solutions in 73% (on average) of the cases and succeeds to reach optimality in 7 instances over 10. However, CPLEX was faster than the mono-phase version for this small problem. In fact, we include all the incompatible columns in CP. So, the CP time increases significantly and CP may select large sets of non disjoint columns as candidates to improve the current solution. This means that strategies that help reducing CP time (like the multi-phase strategy) are needed to efficiently solve set partitioning problems. In three instances, the mono-phase version failed to reach optimality and the error was high. Actually, since the branching implemented is deep only, columns that would possibly be part of a better disjoint solution could be eliminated. A more sophisticated branching may reduce such errors.

Table 4.2 shows results for the multi-phase version of ISUD and CPLEX on 10 instances of the small problem for each level of perturbation. Two new columns are added to this table: the time ISUD reached its optimal solution (Opt.) and the phase in which ISUD finds its best solution (Phase). In this table, we observe that the multi-phase version is faster than CPLEX in most cases (and obviously largely faster than the mono-phase version). It reaches optimal solutions even faster in 9 cases over 10. Actually, it produces solutions with less error on the objective value than the mono-phase version. Interestingly, it presents a higher ratio of disjoint solutions (81%) than the mono-phase version; which means it, more often, finds disjoint solutions without need to branching. Observe that the solution time increases with the perturbation level of the initial solutions.

Figure 4.1 shows how fast each algorithm (mono-phase ISUD, multi-phase ISUD, CPLEX) reaches an optimal solution on instance 1 of the small problem perturbed to 50%. The multi-phase version of ISUD is definitely faster than the mono-phase version. Even if it solves more complementary problems than the mono-phase version, those problems are way smaller. In fact, it makes more iterations with small improvements to the initial solution but in a very

Table 4.1 Mono-phase ISUD vs CPLEX (small instance)

Orig.	Time (sec)			Objective value		*CP* Solutions			
Col. %	ISUD	CPLEX	Ratio %	Init. Err. %	Err. %	Total	Disj.	Max	Avg
50 %	27	5	540	360.6	0	15	13	17	3.9
	24	17	141.2	357.4	0	14	11	13	4.7
	30	13	230.8	347.7	0	17	15	11	3.5
	28	16	175	356.8	202.3	20	9	7	2.7
	25	14	178.6	353.3	0	17	15	13	3.4
	24	16	150	352.8	0	13	11	15	4.8
	32	15	213.3	364.2	0	17	15	9	3.5
	14	17	82.4	368	311.9	10	3	2	2
	28	16	175	360.1	0	15	12	14	4.2
	10	15	66.7	366.9	276.1	8	3	8	4
	24.2	***14.4***	***195.3***	***358.78***	***79.03***	***14.6***	***10.7***	***10.9***	***3.67***

short time. In this example, CPLEX is the fastest. Figure 4.2 represents multi-phase ISUD and CPLEX on instance 4 for the small problem perturbed to 50%. The multi-phase version is faster than CPLEX but not optimal due to the partial branching. This version of ISUD finds many integer solutions that are close to each other. This property is good when using heuristic stopping criteria in large scale problems.

Tables 4.3–4.4 show the same information collected on executions of the multi-phase version of ISUD, which we refer to hereunder by ISUD (without mentioning that is muti-phase), on the medium and large instances. The tables do not show any CPLEX execution because the latter was not able to find any integer solution in a reasonable time. In fact, we tried CPLEX to solve 6 samples (3 instances of the medium problem and 3 of the large one). All the tests were aborted after 10 hours of CPLEX runtime. CPLEX showed an average of 69.55 seconds on the LP relaxation solution time for the medium instances and 592.25 seconds for the large ones. Within the time frame of 10 hours, not even a single integer solution was encountered for all the samples. Interestingly, CP produces relatively small convex combinations with size varying between 2 and 13. These convex combinations are often column-disjoint (89%). The ISUD results are very good for both medium and large instances. When the initial solution is far from the optimal solution, the quality of ISUD solutions decreases and the computation time increases. Developing smart heuristics to find better initial solution will help reducing ISUD time especially for large instances.

The mutli-phase version of ISUD outperforms CPLEX in most test cases. They behave in the same way for small test instances with a slight advantage for ISUD, but for large instances, ISUD solves in few minutes what the CPLEX cannot solve.

We can say that ISUD is particularly useful when the size of the problem is large. As a

Table 4.2 Multi-phase ISUD vs CPLEX (small instance)

Orig.	Time (sec)				Objective value		CP Solutions				
Col. %	ISUD	Opt.	CPLEX	Ratio %	Init. Err. %	Err. %	Phase	Total	Disj.	Max	Avg
50 %	12	6	5	240	360.6	0	4	35	29	6	2.4
	9	4	17	52.9	357.4	0	3	38	31	7	2.4
	9	5	13	69.2	347.7	0	4	33	27	4	2.4
	13	10	15	86.7	356.8	0.1	5	56	39	8	2.9
	9	5	13	69.2	353.3	0	4	33	27	4	2.4
	8	4	16	50	352.8	0	4	33	27	5	2.6
	8	4	14	57.1	364.2	0	4	38	32	5	2.2
	10	6	18	55.6	368	0	4	41	33	9	2.9
	11	7	16	68.8	360.1	0	5	32	26	7	2.5
	10	5	15	66.7	366.9	0	4	36	31	10	2.5
	9.9	***5.6***	***14.2***	***81.62***	***358.78***	***0.01***	***4.1***	***37.5***	***30.2***	***6.5***	***2.52***
35 %	12	8	13	92.3	472.1	0	5	42	33	7	3
	13	8	16	81.2	472.8	0	6	39	29	19	3.4
	9	4	18	50	464.7	0	3	37	31	7	2.6
	10	6	9	111.1	468.4	0	5	35	27	17	3.1
	10	6	14	71.4	463.5	0	4	37	32	5	2.8
	11	8	15	73.3	466.3	0	5	50	39	8	2.8
	16	13	18	88.9	456.3	0	5	43	36	10	3.5
	10	5	15	66.7	471.6	0	3	46	39	5	2.3
	12	8	13	92.3	469.2	0.6	4	46	37	10	3.1
	9	6	7	128.6	464.6	0	5	40	30	7	2.7
	11.2	***7.2***	***13.8***	***85.58***	***466.95***	***0.06***	***4.5***	***41.5***	***33.3***	***9.5***	***2.93***
20 %	12	9	14	85.7	570.2	0	5	44	40	15	3.3
	19	15	18	105.6	561	0	5	53	44	8	3.1
	13	8	19	68.4	559.9	138.2	5	44	23	18	4.3
	14	10	7	200	557.7	0	4	48	43	13	3.3
	18	14	17	105.9	562.5	0	5	52	43	13	3.6
	14	10	14	100	561	0	5	42	33	12	3.4
	12	7	10	120	573.1	0	4	47	40	8	2.8
	16	12	19	84.2	569.6	0	6	51	40	8	3.2
	17	12	17	100	569.3	0	4	63	51	14	3.3
	9	4	13	69.2	573.7	200.6	4	43	31	11	3.5
	14.4	***10.1***	***14.8***	***103.9***	***565.8***	***33.88***	***4.7***	***48.7***	***38.8***	***12***	***3.38***

whole, we conclude the following:

— ISUD reaches optimal solutions most of the time.

— ISUD is faster than CPLEX.

— ISUD is relatively stable compared to CPLEX.

— The quality of ISUD solutions is better for large problems (smaller error).

— The solution time increases with the perturbation level of the initial solutions. It profits from a good heuristic solution if available.

— CP produces relatively small and often disjoint convex combinations with size varying between 2 and 19.

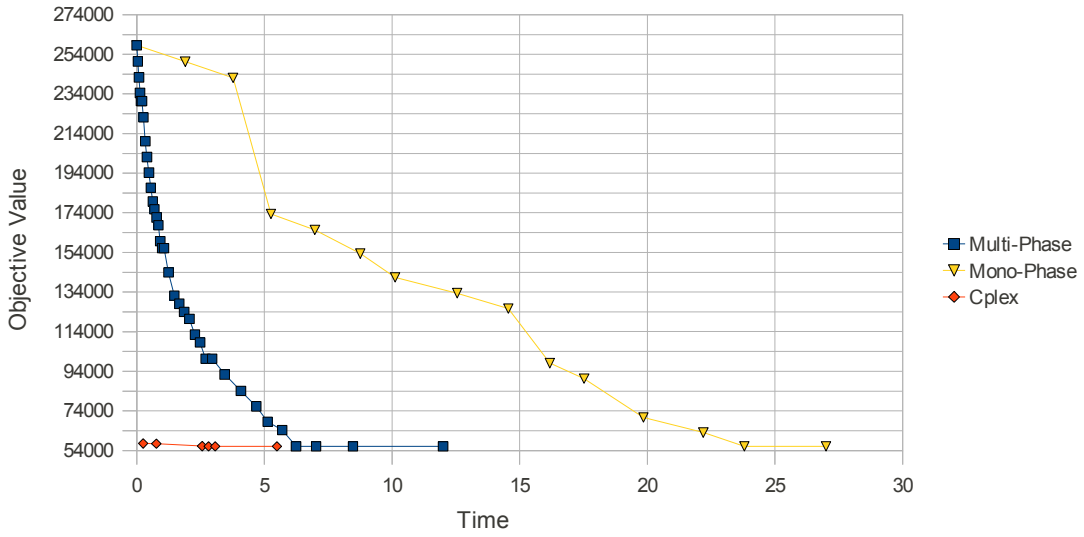

Figure 4.1 Mono-phase ISUD vs Multi-Phase ISUD vs CPLEX (small problem, instance 1, 50%)

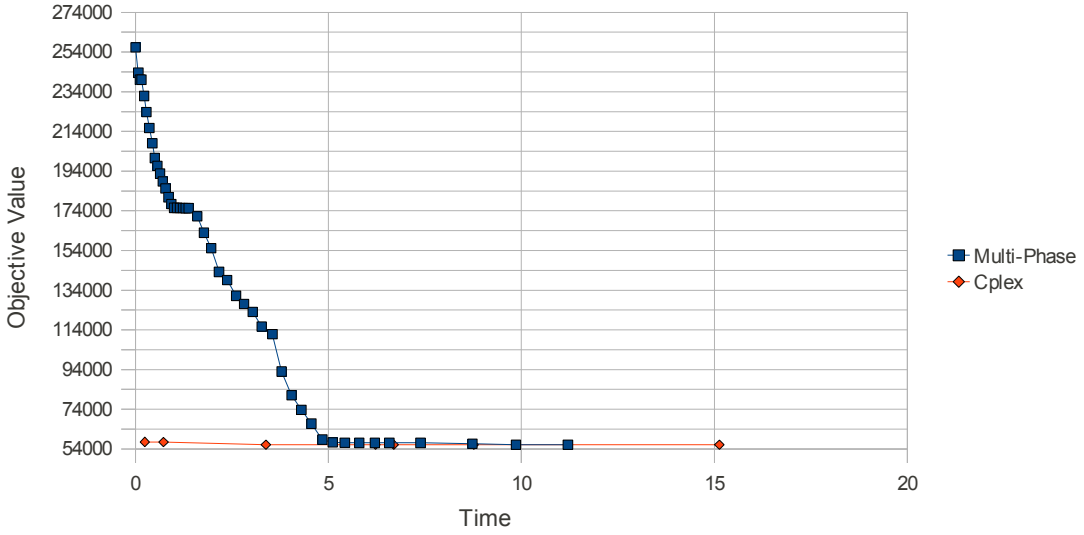

Figure 4.2 Multi-phase ISUD vs CPLEX (small problem, instance 4, 50%)

These good results were obtained on airline crew pairing problems and on bus driver scheduling problems. These are two important domains where set partitioning problems are utilized by the industry. In these domains, it is easy to obtain an initial solution containing good primal information.

Table 4.5 presents some information on the initial solution of the first problem of each group.

Table 4.3 ISUD results for medium instances

Orig.	Time (sec)		Objective value		*CP* Solutions				
Col. %	ISUD	Opt.	Init. Err. %	Err. %	Phase	Total	Disj.	Max	Avg
50 %	59	24	53.43	0	4	15	14	3	2.1
	89	55	51.39	0	5	17	16	3	2.1
	81	51	53.43	0	5	16	15	3	2.2
	126	55	53.44	0	5	15	14	4	2.4
	145	93	51.38	0	6	15	12	6	2.4
	63	32	51.38	0	5	16	15	3	2.1
	165	107	51.38	0	6	18	16	3	2.1
	264	110	51.38	0	6	18	16	3	2.1
	200	102	51.38	0	7	19	15	5	2.3
	57	21	51.38	0	4	15	14	3	2.1
	124.9	***65***	***52.00***	***0***	***5.3***	***16.4***	***14.7***	***3.6***	***2.19***
35 %	221	165	65.77	0	6	24	21	4	2.3
	98	68	65.77	0	5	18	17	4	2.3
	147	80	65.77	10.27	6	19	15	3	2.3
	308	128	67.83	12.33	7	22	17	3	2.2
	360	191	65.77	0	7	23	19	5	2.5
	159	107	67.83	0	6	22	20	6	2.3
	324	222	67.82	0	7	24	20	4	2.4
	114	60	65.77	0	6	16	14	8	2.7
	167	113	65.77	0	6	22	19	5	2.3
	167	110	67.83	0	6	22	20	3	2.2
	206.5	***124.4***	***66.59***	***2.26***	***6.2***	***21.2***	***18.2***	***4.5***	***2.35***
20 %	397	230	82.21	0	8	22	17	8	3.2
	268	201	82.21	0	6	26	24	4	2.6
	142	105	82.21	0	5	25	24	5	2.5
	241	176	82.21	0	6	25	23	5	2.7
	89	64	82.21	41.11	6	15	15	4	2.5
	38	17	82.21	63.72	6	8	8	3	2.1
	383	215	82.21	0	8	27	21	5	2.6
	243	146	82.21	22.61	6	21	18	4	2.6
	298	124	82.21	0	6	23	21	5	2.6
	114	55	82.21	32.88	6	16	15	5	2.4
	221.3	***133.3***	***82.21***	***16.03***	***6.3***	***20.8***	***18.6***	***4.8***	***2.58***

We observe that the percentage good primal information is similar to what is available for reallife crew scheduling problems. The last two columns of Table 4.5 gives the maximum and the average degre of incompatibilities of the columns. These large numbers prove that the same columns were perturbed many times. The number of perturbations grows with the size of the problem. We needed to perturb more before reaching a desirable percentage of unperturbed columns.

We also experiment with a greedy solution for the small problem. This solution has only 29% of good primal information and 37% of the flights are covered with artificial variables. The ISUD improved 46 times the greedy solution, i.e. ISUD found a decreasing sequence

Table 4.4 ISUD results for large instances

Orig.	Time (sec)		Objective value		*CP* Solutions				
Col. %	ISUD	Opt.	Init. Err. %	Err. %	Phase	Total	Disj.	Max	Avg
50 %	996	461	51.88	0	6	18	18	4	2.4
	1173	542	50.34	0	6	19	19	4	2.2
	473	194	50.34	0	5	17	17	4	2.2
	303	198	50.35	13.73	5	15	15	2	2
	1815	520	50.34	0	7	16	16	4	2.5
	392	183	51.87	0	5	19	19	3	2.1
	725	323	50.34	10.68	7	16	13	4	2.3
	1050	376	50.35	0	6	17	17	4	2.2
	918	321	51.87	0	6	16	16	4	2.4
	2887	648	50.34	0	7	19	19	3	2.2
	1073.2	***376.6***	***50.80***	***2.44***	***6***	***17.2***	***16.9***	***3.6***	***2.25***
35 %	1258	632	65.60	0	6	25	25	3	2.2
	1166	557	65.60	0	6	23	23	5	2.5
	1102	438	65.59	0	6	24	24	5	2.3
	2655	765	65.60	0	7	26	26	5	2.3
	623	351	65.60	15.26	6	23	21	5	2.3
	2061	813	65.60	0	7	24	24	5	2.4
	460	268	65.60	19.83	5	18	17	5	2.5
	517	419	65.60	6.10	5	24	24	5	2.3
	2691	1191	65.59	0	8	27	25	4	2.6
	928	436	65.60	10.68	6	23	22	7	2.5
	1346.1	***587***	***65.60***	***5.19***	***6.2***	***23.7***	***23.1***	***4.9***	***2.39***
20 %	2806	1402	82.38	0	7	27	27	7	3.4
	2968	1194	80.85	0	8	24	22	13	3.7
	2472	1029	80.85	0	7	31	30	7	2.5
	2363	1120	80.86	0	8	23	21	13	3.4
	3556	1981	80.86	0	8	30	27	8	3.3
	1553	1076	80.86	0	6	32	32	6	2.6
	229	138	80.85	56.45	6	12	12	4	2.3
	3293	1845	80.85	0	7	31	31	8	3.2
	2008	842	80.86	0	8	21	19	10	3.6
	2923	1247	80.86	0	7	34	33	4	2.6
	2417.1	***1187.4***	***81.01***	***5.64***	***7.2***	***26.5***	***25.4***	***8***	***3.06***

of 46 integer solutions, before stopping and reduced to 7% the number of flights covered by artificial variables. The mean cost per column (excluding the artificial ones) was 35% below in the greedy than the mean cost per column in the optimal solution. The greedy solution contains columns good with regards to the dual feasibility but very poor in primal information. It is the opposite of what ISUD likes to exploit.

It is also possible to have good primal information for vehicle, crew and many other personnel scheduling problems when we reoptimize a planned solution after some perturbations. A large part of the planned solution will remain in the reoptimized solution. A similar situation

Table 4.5 Information on initial solutions

Problem	Orig. Col. %	% of good primal information	Max. deg. of incomp.	Avg. deg. of incompatibility
Small (airline)	50	88	7	1.2
	35	82	9	2
	20	73	12	3
Medium (Bus)	50	94	17	2.6
	35	90	17	4.5
	20	88	17	5.3
Large (Bus)	50	91	12	2.2
	35	83	15	4.1
	20	73	22	6.8

appears in the two stages stochastic programming solved with the L-shaped method. For each scenario, a subproblem updates the planned solution according to the modified data. A large part of the planned solution remains in the solution of each subproblem.

Even if the conclusion of the experimentation cannot yet be generalized to all set partitioning problems, it is already appliable to a wide class of problems very important in practice. For the other problems, one could see how to adapt the multi-phase strategy by modifying the way we compute the degree of incompatibilities.

4.7 Conclusion

We introduce a constructive method that finds a decreasing sequence of integer solutions to a set partitioning problem by decomposing it into a reduced problem RP and a complementary problem CP.

The optimization of RP with the primal simplex involves carrying out pivots on variables that move from one integer solution to a better one. When the RP has no more improving pivots, the CP identifies a group of variables producing a sequence of pivots moving to a better integer solution after some degenerate pivots. Iterations on RP and CP permit to reach an optimal integer solution by only using normal pivots on positive coefficients.

A first implementation of the algorithm with only a partial branching procedure produces optimal solutions in many cases and near optimal solutions in the majority of the other cases.

On large scale problems, the very good solutions are obtained in a small fraction of the time to obtain same quality integer solutions with the LP relaxation and the branch and bound approach. In fact, when we relax the disjoint columns condition, the CP becomes a linear program to solve. This relaxed problem produces disjoint columns most of the time and permits to rapidly improve the integer solution. This opens up a new way to obtain very good integer solutions for large set partitioning problems.

Future research on branching strategies and/or cutting plane methods should be able to close the small optimality gap and produce a new optimal solution method. Many good branching/cutting methods have been proposed in the literature for the set partitioning problem. Some of these methods could probably be adapted to the $RP - CP$ decomposition of ISUD. Also, combining ISUD and metaheuristics to find a good initial solution and to well populate CP could reduce significantly the solution time and produce solutions with higher quality.

e-companion: Construction of a sequence of basic solutions of $\mathbb{P}$ using only ordinary simplex pivots

The results presented in this e-companion are not used in the algorithm. They are presented here working directly with $\mathbb{P}$ to place our theoretical results in the same context as those of Balas and Padberg, to show the improvements obtained.

In Sections 4.3 and 4.5 we show that we can move from an integer solution to $\mathbb{P}$ to an optimal solution by performing ordinary simplex pivots on RP and CP. This section improves the results of Balas and Padberg on the sequence of adjacent basic solutions of the problem $\mathbb{P}$ permitting us to move from an initial integer solution to an optimal integer solution. It transfers to the problem $\mathbb{P}$ the sequence of solutions found by working with RP and CP. It moreover shows that this sequence has more interesting properties than the sequence of Balas and Padberg, which uses degenerate pivots on the negative elements of the constraint matrix. This difference is one of the elements that contributes to the effectiveness of the new method while restricting the search domain of the sets of variables that must be entered into the basis simultaneously to improve the integer solution. It thus becomes possible to carry out a large part of the search of these sets with the ordinary simplex algorithm. The use of combinatorial exploration is still necessary, but it often represents only a small fraction of the solution time.

Let us consider the problem $\mathbb{P}$ to which we add artificial variables $y_i, i = 1, \ldots, m$ with cost M as a simplex phase I.

Proposition (EC.1)**.** *Let x^1 be a nonoptimal integer solution to $\mathbb{P}$ and P the index set of its positive components. Let CP contain the $m - p$ degenerate constraints for the solution x^1 ($p = |P|$). Then there exists x^*_I, a column-disjoint solution with a negative reduced cost. Let S be the set of indices $j \in I$ such that $x^*_I > 0$. Starting from x^1 with the associated basis containing the variables x^1_j, $j \in P$ and the artificial variables y_i, $i \in N$, the primal simplex algorithm reaches an improved integer solution x^{**} in $|S|$ pivots. Moreover, it uses ordinary simplex pivots on a column of negative reduced cost and on a row of minimal ratio* $\min_i\{\bar{b}_i/\bar{A}^i_j | \bar{A}^i_j > 0\}$.

Proof. By Proposition 4.3.12 there exists x^*_I, a column-disjoint solution to CP with a negative marginal cost. If the set S of this solution is not minimal we replace it by a subset that is minimal.. The value of this solution is $x^*_j = \frac{1}{|S|}$, $j \in S$ and $x^*_j = 0$, $j \notin S$.

We now establish results on the linear independence of the columns. The columns $A_j, j \in S$ are linearly independent because they are disjoint. If $S = \{j\}$ the variable x_j is compatible because $\bar{A}^N_j = 0$ and A_j is dependent on the columns $A_j, j \in P$. If $|S| > 1$ the variables

$x_j, j \in S$ are incompatible and $\bar{A}_j^N \neq 0$. If we had $j \in S$ compatible then $\{j\}$ and $S - \{j\}$ would define two solutions of CP and S would not be minimal. If $|S| > 1$, each column $A_j, j \in S$ is linearly independent of the columns $A_j, j \in P$ because $\bar{A}_j^N = 0, j \in P$ and $\bar{A}_j^N \neq 0, j \in P$.

Note that these results are valid for the columns of A or $\bar{A}$ because we move from one to the other with a nonsingular transformation B^{-1}.

We now discuss on which element to carry out the first pivot.

$$0 > \sum_{j \in S} \bar{c}_j \cdot x_j^* = \frac{1}{|S|} \sum_{j \in S} \bar{c}_j$$

because the reduced cost of x^* is negative and $x_j^* = \frac{1}{|S|}, j \in S$.

There exists $j_1 \in S$ such that $\bar{c}_{j_1} < 0$, otherwise $\sum_{j \in S} \bar{c}_j \geq 0$. For the case of $|S| = 1$ the first pivot corresponds to the $|S|^{\text{th}}$ pivot and is considered later. If $|S| > 1$, there exists $k \in N$ such that $\bar{A}_{j_1}^k \neq 0$ because $j_1 \in S \subseteq I$ is an incompatible column. There exists $k_1 \in N$ such that $\bar{A}_{j_1}^{k_1} > 0$ because

$$\bar{c}_{j_1} = c_{j_1} - \sum_{i=1}^{m} \bar{A}_{j_1}^i c_i = c_j - \sum_{i \in P} \bar{A}_{j_1}^i c_i - \sum_{i \in N} \bar{A}_{j_1}^i M < 0$$

and one of the $\bar{A}_{j_1}^i, i \in N$ must be positive. If they were all negative the reduced cost would be positive because M is much larger than the values of $c_i, i \in P$. The pivot can be carried out on row k_1 because

$$0 = \bar{b}^{k_1} / \bar{A}_{j_1}^{k_1} = \min_k \{\bar{b}^k / \bar{A}_{j_1}^k | \bar{A}_{j_1}^k > 0\}.$$

Indeed, $\bar{b}^{k_1} = 0$ and $\bar{b}^k > 0, k \in P$. This pivot is degenerate because the ratio is zero. The basic variable associated with row k_1 is an artificial variable. The variable $j \in S$ enters the basis and the artificial variable leaves.

$\bar{A}$ and $\bar{c}$ hereafter represent the modified values after the basis update. For the case of $|S| = 2$ the second pivot corresponds to the $|S|^{\text{th}}$ pivot and is considered later. If $|S| > 2$, there exists $j_2 \in S - \{j_1\}$ such that $\bar{c}_{j_2} < 0$ because the solution x^1 can be improved with the variables in $S - \{j_1\}$. There exists $k \in N - \{k_1\}$ such that $\bar{A}_{j_2}^k \neq 0$ because $\bar{A}_{j_2}^N$ is linearly independent of the $\bar{A}_j, j \in P$ and of $\bar{A}_{j_1}^N$. There exists $k_2 \in N - \{k_1\}$ such that $\bar{A}_{j_2}^{k_2} > 0$ because of the negative reduced cost and we can carry out a degenerate pivot in which the variable $j_2 \in S$ enters the basis and k_2 leaves.

This sequence of degenerate pivots can continue for $|S| - 1$ pivots. The $|S|^{\text{th}}$ pivot is different. The reduced cost $\bar{c}_j$ of the last element $j_{|S|}$ of S is again negative because the solution x_1 has

not yet been modified and it can be modified with the variable $x_{j_{|S|}}$

This nondegenerate pivot is not carried out on a row $k \in N$ because their $\bar{b}^k$ are zero. It is carried out on a row $k \in P$ where $\bar{b}^k = b^k = 1$. Since the variable $x_{j_{|S|}}$ takes the value 1 in x^{**} the ratio $\bar{b}^k / \bar{A}^k_{j_{|S|}} = 1 > 0$. □

Corollary (EC.1)**.** *The solution x^{**} is adjacent to x^1.*

Proof. x is an adjacent solution to x^1 because it can be reached by performing a single nondegenerate pivot. □

Proposition (EC.2)**.** *Let x^1 be a nonoptimal integer solution to $\mathbb{P}$ and P the index set of its positive components. Starting from x^1 with the associated basis containing the variables x^1_j, $j \in P$ and artificial variables $y_i, i \in N$ to complete the basis, there exists a sequence of simplex pivots that reaches an optimal solution. The sequence includes two types of pivots:*

1. *Ordinary pivots entering into the basis a variable x_j with $\bar{c}_j < 0$ via a pivot on $\bar{A}^i_j > 0$.*
2. *Degenerate pivots replacing a variable $x_j = 0$ by an artificial variable $y_i = 0$.*

Proof. The construction of the sequence starts with the solution of CP defined from x^1 in order to obtain a column-disjoint solution with a negative reduced cost having a minimal set S^1 of nonzero variables.

The $|S^1|$ first terms of the sequence are the ordinary pivots identified in Proposition EC.1. We reach x^2, a new integer solution that is adjacent to x^1. If x^2 is optimal the sequence is terminated.

Otherwise during the $|S^1|^{\text{th}}$ pivot more than one x_j variable can be cancelled out. The basic variables associated with x^2 can be partitioned into three sets:

$$B^2_1 = \{j | x^2_j = 1\}, \quad B^2_0 = \{j | x^2_j = 0\}, \quad \text{and} \quad N^2 = \{i | y_i = 0\}.$$

We thus carry out $|B^2_0|$ degenerate pivots, replacing the basic variable $x^2_j, j \in B^2_0$ by the artificial variable y_i associated with the same constraint. These pivots are easy to carry out because we know the pairs of variables that enter and leave the basis. Moreover, the basis update is easy because the entering variables have columns from the identity matrix.

We thus have again the conditions for the application of Proposition EC.1. We have x^2, a nonoptimal integer solution, a basis formed from $x_j = 1, j \in P^2 = B^2_0$ and artificial variables $y_i, i \in N^2$. We can then identify with CP a set S^2 of variables to enter into the basis to obtain x^3, a better integer solution. The set adds $|S^2|$ ordinary pivots to the sequence. This process is continued until an optimal solution is obtained. □

The sequence of basic solutions proposed here is different from that of Balas and Padberg in the following aspects:

- All the pivots are carried out on the $\bar{A}^i_j > 0$.
- There are no zero basic variables that take the value 1 during a pivot (i..e. $Q^- = \emptyset$).
- There are degenerate pivots that are easy to identify and carry out; they replace the zero variables in the basis by artificial variables.
- The sequence of pivots can be identified even if we do not know the optimal solution in advance.
- The group of pivots to carry out to move to an adjacent integer solution is identified by solving a complementary problem. Branching is not always necessary to obtain the desired type of solution.

CHAPITRE 5 ARTICLE 2 : IMPROVING SET PARTITIONING PROBLEM SOLUTIONS BY ZOOMING AROUND AN IMPROVING DIRECTION

Improving set partitioning problem solutions by zooming around an improving direction

Abdelouahab Zaghrouti, Issmail El Hallaoui, François Soumis

Article soumis à European Journal of Operational Research

5.1 Introduction

Consider the set partitioning problem (SPP):

$$\begin{array}{ll} \min & cx \\ (\mathbb{P}) & Ax = e \\ & x_j \quad \text{binary}, \quad j \in \{1, \ldots, n\}, \end{array}$$

where A is an $m \times n$ matrix with binary coefficients, c is an arbitrary vector with nonnegative integer components of dimension n, and $e = (1, \ldots, 1)$ is a vector of dimension dictated by the context. We assume that $\mathbb{P}$ is feasible and A is of full rank. When we relax the binary constraints, we obtain the continuous relaxation denoted by LP.

5.1.1 Literature review

The SPP has been widely studied in the last four decades, mainly because of its many applications in industry. A partial list of these applications includes truck deliveries (Balinski & Quandt (1964), Cullen et al. (1981)), vehicle routing (Desrochers et al. (1992)), bus driver scheduling (Desrochers & Soumis (1989)), airline crew scheduling (Hoffman & Padberg (1993), Gamache et al. (1999), Barnhart et al. (1998)), and simultaneous locomotive and car assignment (Cordeau et al. (2001)). Several companies provide commercial optimizers to these problems using this mathematical model or one of its variants.

The literature on the SPP is abundant (see the survey by Balas & Padberg (1976)). As is the case for generic integer linear programs, there are three main classes of algorithms for SPPs (Letchford & Lodi (2002)): *dual fractional*, *dual integral*, and *primal methods*. Dual fractional algorithms maintain optimality and linear constraints feasibility (i.e. constraints $Ax = e$) at every iteration, and they stop when integrality is achieved. They are typically standard cutting plane procedures such as Gomory's algorithm (Gomory (1958)). The classical branch-and-bound scheme is also based on a dual fractional approach, in particular for the determination of lower bounds. SPP is usually solved, especially when columns are not known a priori, by branch and price (and cut) (Barnhart et al. (1998); Lübbecke & Desrosiers (2005)) where each node of the branch-and-bound tree is solved by column generation. The classical approach uses the simplex algorithm or an interior point method (such as the CPLEX barrier approach) to solve the linear relaxation of $\mathbb{P}$ to find a lower bound, often resorting to perturbation methods to escape the degeneracy inherent to this problem. These algorithms often provide solutions that are very fractional, i.e., infeasible from the

integrality point of view. This dual (or *dual fractional* as it is called in Letchford & Lodi (2002)) approach strives to improve the lower bound using branching methods or cuts until a good integer solution is found. This approach is effective for small problems and remains attractive for problems of a moderate size. For large problems, a very fractional solution leads to a large branching tree, and we must often stop the solution process without finding a good integer solution. Dual integral methods, not much practical, maintain integrality and optimality, and they terminate when the primal linear constraints are satisfied. An example is the algorithm developed by Gomory (1963).

Finally, primal algorithms maintain feasibility (and integrality) throughout the process, and they stop when optimality is reached. Several authors have investigated ways to find a nonincreasing sequence of basic integer solutions leading to an optimal solution. The existence of such a sequence was proved in Balas & Padberg (1972, 1975). The proof relies on the quasi-integrality of SPPs, i.e., every edge of the convex hull of $\mathbb{P}$ is also an edge of LP. The proposed algorithms (Haus et al. (2001); Rönnberg & Larsson (2009); Thompson (2002)) explore by enumeration the tree of adjacent degenerate bases associated with a given extreme point to find a basis permitting a nondegenerate pivot that improves the solution. These highly combinatorial methods are not effective for large problems, mainly because of the severe degeneracy. The number of adjacent degenerate bases associated with a given extreme point can be very huge, actually.

Zaghrouti et al. (2014) introduce an efficient algorithm, the integral simplex using decomposition (ISUD), that can find the terms of the sequence without suffering from degeneracy. ISUD is a primal approach that moves from an integer solution to an adjacent one until optimality is reached. ISUD decomposes the original problem into a reduced problem (RP) and a complementary problem (CP) that are easier to solve. We solve RP to find an improved integer solution in the vector subspace generated by columns of the current integer solution, i.e. columns corresponding to nondegenerate variables (that value 1). Note that a pivot on a negative-reduced-cost variable in this vector subspace decreases the objective value and more importantly the integrality is preserved in a straightforward manner as proved in Zaghrouti et al. (2014). We solve then CP to find an integer descent direction (i.e., leading to an improved integer solution) in the complementary vector subspace. Integrality is handled in both RP and CP but mainly in CP, which finds integer directions. Once the direction has been identified by CP, RP uses it to update the current integer solution and we iterate until an optimal solution is reached. ISUD is more efficient than the conventional dual (fractional) approach on pure SPPs with some special structure, even though the conventional approach has been much improved since its introduction 40 years ago.

5.1.2 Main contributions

In this paper, we introduce a general framework for *vector space decompositions*, or simply called here RP-CP decompositions, that decompose the set partitioning problem into an RP and a CP. We show that ISUD developed in Zaghrouti et al. (2014) uses a particular decomposition, in which integrality is handled in both RP and CP, to find a decreasing sequence of integer solutions leading to an optimal solution. We introduce a new algorithm using a dynamic decomposition where, in contrast to ISUD, integrality is handled only in RP, and the CP is only used to provide descent directions, needed to update the decomposition. When CP finds a fractional descent direction, instead of using cuts (Rosat et al. (2014)) or branching (Zaghrouti et al. (2014)) to find an integer direction, we use this descent direction to indicate an area of potential improvement. The new algorithm improves, at each iteration, the current integer solution by solving a very small RP that we define by zooming around the descent direction. The RPs solved are tens times smaller than the original problem. As will be explained in more detail later in the paper, we use reduced costs (expressed as the objective function of CP) to find descent directions, i.e. to guide the search, and construct RPs (neighborhoods) via a computable distance to the vector subspace generated by the columns of the current integer solution. The neighborhoods thus constructed are likely to contain *improved integer* solutions even when the distance is small. Some techniques for finding such neighborhoods are discussed.

Our ultimate goal is to increase the success rate of reaching an optimal or near optimal solution on very hard instances, without increasing the solution time, via local improvements to the current integer solution. Local improvement of solutions to very large problems is highly desirable in practice. This will allow to solve large industrial SPPs within an exact primal paradigm. The zooming algorithm is globally primal (it moves through a decreasing sequence of integer solutions to the optimal solution of $\mathbb{P}$) but locally dual fractional (when necessary it solves a small (local) MIP using the dual fractional approach).

In Section 5.2, we present a general framework for RP–CP decompositions and discuss the theoretical foundations of this framework. We briefly present ISUD and discuss its strengths and weaknesses in Section 5.3, and we show that it uses a particular RP–CP decomposition. We discuss the zooming approach and its advantages in Section 5.4. We evaluate the algorithm on large vehicle and crew scheduling problems, and our numerical results are in Section 5.5. We show in this section that the zooming algorithm significantly improves the integral simplex algorithm of Zaghrouti et al. (2014) and works better on set partitioning instances from the transportation industry, for which it rapidly reaches optimal or near-optimal solutions to instances very hard to both ISUD and CPLEX. In Section 5.6 we discuss possible

extensions of the approach.

5.2 RP–CP decomposition

Let $\mathcal{Q}$ be an index subset of linearly independent columns of A, containing at least indices of columns corresponding to positive variables in a given solution to $\mathbb{P}$. We partition the columns of A into two groups, those that are compatible with $\mathcal{Q}$ and those that are not, using the following definition:

Definition 5.2.1. *A column (and its corresponding variable) or a convex combination of columns is said to be compatible with $\mathcal{Q}$ if it can be written as a linear combination of columns indexed in $\mathcal{Q}$. Otherwise, it is said incompatible.*

The index set of compatible columns is denoted $c_{\mathcal{Q}}$ and the index set of incompatible columns is denoted $\mathcal{I}_{\mathcal{Q}}$. Clearly $\mathcal{Q} \subseteq c_{\mathcal{Q}}$ and $|\mathcal{Q}| \leq m$. $\mathcal{Q}$ is said to be *nondegenerate* if it is restricted to columns corresponding to positive variables (in a given solution); otherwise it is said *degenerate*. Let $A_{\mathcal{Q}} = \begin{pmatrix} A^1_{\mathcal{Q}} \\ A^2_{\mathcal{Q}} \end{pmatrix}$ be a submatrix of A composed of columns indexed in $\mathcal{Q}$ where $A^1_{\mathcal{Q}}$ is, without loss of generality, composed of the first $|\mathcal{Q}|$ linearly independent rows. $A^2_{\mathcal{Q}}$ is of course composed of dependent rows. Similarily, $A_{c_{\mathcal{Q}}} = \begin{pmatrix} A^1_{c_{\mathcal{Q}}} \\ A^2_{c_{\mathcal{Q}}} \end{pmatrix}$ (resp. $A_{\mathcal{I}_{\mathcal{Q}}} = \begin{pmatrix} A^1_{\mathcal{I}_{\mathcal{Q}}} \\ A^2_{\mathcal{I}_{\mathcal{Q}}} \end{pmatrix}$) is a submatrix of A composed of columns indexed in $c_{\mathcal{Q}}$ (resp. $\mathcal{I}_{\mathcal{Q}}$) where $A^1_{c_{\mathcal{Q}}}$ and $A^1_{\mathcal{I}_{\mathcal{Q}}}$ are also composed of the same first $|\mathcal{Q}|$ rows as in $A^1_{\mathcal{Q}}$.

When we consider only compatible columns $c_{\mathcal{Q}}$ and therefore the first $|\mathcal{Q}|$ rows, we obtain the RP, which can be formulated as

$$(RP_{\mathcal{Q}}) \quad \min_{x_{c_{\mathcal{Q}}}} \quad c_{c_{\mathcal{Q}}} \cdot x_{c_{\mathcal{Q}}} \tag{5.1}$$

$$\text{s.t.} \quad A^1_{c_{\mathcal{Q}}} x_{c_{\mathcal{Q}}} = e \tag{5.2}$$

$$x_j \ \text{ binary}, j \in c_{\mathcal{Q}} \tag{5.3}$$

where $c_{c_{\mathcal{Q}}}$ is the subvector of the compatible variables costs and $x_{c_{\mathcal{Q}}}$ is the subvector of compatible variables. $RP_{\mathcal{Q}}$ is therefore an SPP restricted to the compatible variables and the first $|\mathcal{Q}|$ rows. Observe that its columns are in $\mathbb{R}^{|\mathcal{Q}|}$ (a reduced dimension). It is important to stress the fact that each solution to $RP_{\mathcal{Q}}$ can be completed by zeros to form a solution to $\mathbb{P}$.

The CP (containing the incompatible variables) is formulated as follows:

$$(CP_{\mathcal{Q}}) \qquad z^{CP}_{\mathcal{Q}} = \min_{v,\lambda} \quad \sum_{j\in I_{\mathcal{Q}}} c_j v_j - \sum_{l\in\mathcal{Q}} c_l \lambda_l \tag{5.4}$$

$$\text{s.t.} \qquad \sum_{j\in I_{\mathcal{Q}}} v_j A_j - \sum_{l\in\mathcal{Q}} \lambda_l A_l = 0 \tag{5.5}$$

$$e \cdot v = 1 \tag{5.6}$$

$$v \geq 0, \lambda \in \mathbb{R}^{|\mathcal{Q}|}. \tag{5.7}$$

The positive v_j variables indicate entering variables and the positive λ_l variables indicate exiting variables. We look for a group of entering variables that will replace the exiting variables. Of course, the cost difference (5.4) between the entering and exiting variables (the reduced cost) must be negative for a minimization problem in order to improve the objective value. In other words, we look, by imposing constraints (5.5), for a convex combination, with a negative reduced cost, of incompatible columns A_j (of the constraint matrix A) that is compatible with $\mathcal{Q}$ according to Definition 5.2.1, i.e. that is a linear combination of columns A_l indexed in $\mathcal{Q}$. Without constraint (5.6), the feasible domain of $CP_{\mathcal{Q}}$ is an unbounded cone and the problem is unbounded. With this constraint, the problem is bounded and provides a normalized improving direction. It is therefore called a *normalization constraint*. To guarantee feasibility, we add an artificial variable that costs 0 and only contributes to the normalization constraint (by 1); this way, $CP_{\mathcal{Q}}$ is feasible and $z^{CP}_{\mathcal{Q}} \leq 0$.

Since the variables λ_l, linked to variables v_j via constraints (5.5), are associated with linearly independent columns A_l, $l \in \mathcal{Q}$, they can be substituted by the variables v_j, $j \in I_{\mathcal{Q}}$. Actually, we have $\lambda = (A^1_{\mathcal{Q}})^{-1} A^1_{\mathcal{I}_{\mathcal{Q}}} v$. After the substitution, the CP can be rewritten in the following equivalent form:

$$(CP_{\mathcal{Q}}) \qquad z^{CP}_{\mathcal{Q}} = \min_{v} \quad \tilde{c} \cdot v \tag{5.8}$$

$$\text{s.t.} \qquad M A_{I_{\mathcal{Q}}} v = 0 \tag{5.9}$$

$$\text{Constraints (5.6)–(5.7)} \tag{5.10}$$

where $v \in \mathbb{R}^{|\mathcal{I}_{\mathcal{Q}}|}$, $\tilde{c} = \left(c^\top_{\mathcal{I}_{\mathcal{Q}}} - c^\top_{\mathcal{Q}} \left(A^1_{\mathcal{Q}}\right)^{-1} A^1_{\mathcal{I}_{\mathcal{Q}}}\right)$ is the vector of reduced costs with respect to the constraints of the RP and $M = \left(A^2_{\mathcal{Q}} \left(A^1_{\mathcal{Q}}\right)^{-1}, -I_{m-|\mathcal{Q}|}\right)$ is a projection matrix, called the *compatibility* matrix, on the complementary vector subspace. $c_{\mathcal{I}_{\mathcal{Q}}}$ is the subvector of incompatible variables costs and $I_{m-|\mathcal{Q}|}$ is the identity matrix of dimension $m - |\mathcal{Q}|$. With the normalization constraint, the dimension of the columns of the CP is $m - |\mathcal{Q}| + 1$, i.e., the

dimension of the complementary vector subspace plus 1.

If we relax the binary constraints (5.3), we obtain the improved primal simplex decomposition introduced in Elhallaoui et al. (2011). In that paper, the authors proved that if $\mathcal{Q}$ is nondegenerate; a pivot on any negative-reduced-cost compatible variable or a sequence of pivots on the set of the entering variables (i.e., $v_j > 0$) improves (strictly) the current solution of LP *iff* $z^{CP}_{\mathcal{Q}} < 0$.

Let s be the current integer solution to $\mathbb{P}$ and $s_{CP_{\mathcal{Q}}}$ an optimal solution to $CP_{\mathcal{Q}}$. For the sake of simplicity, $s_{CP_{\mathcal{Q}}}$ is completed by $|\mathcal{C}_{\mathcal{Q}} \setminus \mathcal{Q}|$ zeros (corresponding to compatible variables that are absent from $CP_{\mathcal{Q}}$) to have the same dimension as s. A direction $d = \rho s_{CP_{\mathcal{Q}}}$ is a descent direction associated with $s_{CP_{\mathcal{Q}}}$ if there exists $\rho > 0$ such that $s' = s + d$ is an extreme point of LP and $z^{CP}_{\mathcal{Q}} < 0$.

Definition 5.2.2. *$s_{CP_{\mathcal{Q}}}$ and its associated descent direction d are said to be disjoint if the columns $\{A_j | v_j > 0, j \in I_{\mathcal{Q}}\}$ are pairwise row-disjoint. They are said to be integer if $s + d$ is an improved integer solution and, fractional otherwise.*

The following proposition provides a general characterization of the integrality of descent directions. Let $\mathcal{Q}_s$ be the set of indices of columns coresponding to positive variables of the current integer solution s within $\mathcal{Q}$.

Proposition 5.2.3. *$s_{CP_{\mathcal{Q}}}$ is integer if and only if*

1. *$\{A_j | v_j > 0, j \in I_{\mathcal{Q}}\} \cup \{A_l | \lambda_l < 0, l \in \mathcal{Q}\}$ is a set of pairwise row-disjoint columns.*
2. *$\{l | \lambda_l > 0, l \in \mathcal{Q}\} \subset \mathcal{Q}_s$.*
3. *$\{l | \lambda_l < 0, l \in \mathcal{Q}\} \subset \mathcal{Q} \setminus \mathcal{Q}_s$.*

Proof. First ($\Longrightarrow$), if the direction found by $CP_{\mathcal{Q}}$ is integer, this means that there exists a $\rho > 0$ such that $s' = s + d$ is an integer solution, with $d = \rho\delta$ and $\delta = s_{CP_{\mathcal{Q}}}$. For each j, we have $\delta_j = \frac{s'_j - s_j}{\rho}$. As s and s' are 0-1 vectors, we get:

- $s'_j = s_j \iff \delta_j = 0$
- $s'_j = 1, s_j = 0 \iff \delta_j = \frac{1}{\rho}$
- $s'_j = 0, s_j = 1 \iff \delta_j = \frac{-1}{\rho}$

$\sum_{j \in I_{\mathcal{Q}}} v_j A_j - \sum_{l \in \mathcal{Q}} \lambda_l A_l = 0$. So, $\delta_j = v_j$ for $j \in I_{\mathcal{Q}}$ and $\delta_l = -\lambda_l$ for $l \in \mathcal{Q}$. We are interested in nonzero components (support) of δ. Remark that $\lambda_l > 0$ ($\delta_j = -\lambda_l = \frac{-1}{\rho}$) is equivalent to $s_j = 1$ and $s'_j = 0$ which means that $\{l | \lambda_l > 0, l \in \mathcal{Q}\} \subset \mathcal{Q}_s$. Observe also that $\lambda_l < 0$ ($\delta_j = -\lambda_l = \frac{1}{\rho}$) is equivalent to $s_j = 0$ and $s'_j = 1$. That means that $\{l | \lambda_l < 0, l \in \mathcal{Q}\} \subset \mathcal{Q} \setminus \mathcal{Q}_s$. We can rewrite (5.5) as

$$\sum_{j : j \in \mathcal{I}_{\mathcal{Q}}, v_j > 0} A_j + \sum_{l : \lambda_l < 0} A_l = \sum_{l : \lambda_l > 0} A_l$$

From above, we can conclude that $\{A_j | v_j > 0, j \in \mathcal{I}_{\mathcal{Q}}\} \cup \{A_l | \lambda_l < 0, l \in \mathcal{Q}\}$ is a set of pairwise row-disjoint columns.

Second, to prove the left implication ($\Longleftarrow$), we know that:

$$\sum_{j|j\in\mathcal{I}_{\mathcal{Q}},v_j>0} v_j A_j - \sum_{l|\lambda_l<0} \lambda_l A_l = \sum_{l|\lambda_l>0} \lambda_l A_l$$

Observe that both the columns of the right and left sides are pairwise row-disjoint and consequently they are linearly independent. We can show that there exists a certain positive ρ such that $v_j = \frac{1}{\rho}$, $\lambda_l = \frac{-1}{\rho}$ when $\lambda_l < 0$ and $\lambda_l = \frac{1}{\rho}$ when $\lambda_l > 0$ is a unique solution (involving columns A_j corresponding to nonzero variables v_j and λ_l) to the linear system just above. So, the solution s' obtained by replacing in s the columns of the right hand side by the columns of the left hand side is an improved integer solution because the columns are disjoint and the cost difference between the two solutions is negative. □

This proposition is a generalization of a theoretical result of i) Zaghrouti et al. (2014) showing that if $\mathcal{Q}$ is nondegenerate (i.e. $\mathcal{Q} = \mathcal{Q}_s$) and the solution to CP is disjoint then we obtain an integer descent direction, and that of ii) Balas & Padberg (1975) considering that $\mathcal{Q}$ contains indices of all basic columns. A disjoint $s_{CP_{\mathcal{Q}}}$ does not guarantee that s' is integer in the general case when $\mathcal{Q}$ is degenerate.

Remark 5.2.4. *Observe also that:*

- *The formulation (5.8)–(5.10) of $CP_{\mathcal{Q}}$ can be strengthened by imposing that $\lambda_l \geq 0, l \in \mathcal{Q}_s$ and $\lambda_l \leq 0, l \in \mathcal{Q} \setminus \mathcal{Q}_s$ where $\lambda = (A^1_{\mathcal{Q}})^{-1} A^1_{\mathcal{I}_{\mathcal{Q}}} v$.*
- *The strenghtened formulation is still bounded and the improved solution s' is not necessarily adjacent to s.*

The following proposition proves the existence of an optimal RP–CP decomposition, in the sense that instead of solving the original problem, we may solve smaller problems (the RP and CP) and obtain an optimal solution to the original problem. This theoretically motivates further our research and provides in some sense a mathematical foundation for the proposed approach.

Proposition 5.2.5. *There exists an optimal decomposition RP–CP such that an optimal solution to $RP_{\mathcal{Q}}$ is also optimal to $\mathbb{P}$ and $z^{CP}_{\mathcal{Q}}$, the objective value of $CP_{\mathcal{Q}}$, is nonnegative.*

Proof. Consider in $\mathcal{Q}$ all columns that correspond to positive variables of the optimal solution and in RP all columns that are involved in fractional descent directions. It is sufficient (at

least theoretically) to add a finite number of facets to the RP to cut the fractional solutions. Obviously, the CP cannot find, after adding these cuts, a descent direction like in Rosat et al. (2014), so $z_{\mathcal{Q}}^{CP} \geq 0$. □

This decomposition allows handling the integrality constraints in the RP instead of handling them, as ISUD does, in the CP. In the worst case, the RP may coincide with the original problem. In practice, large SPPs are generally highly degenerate, and the RP is significantly smaller than the original problem as a result of this inherent degeneracy.

Below, we present two propositions that help proving the exactness of the proposed approach and assessing the solution quality. To prove these propositions, we need the following lemma. We suppose that $RP_{\mathcal{Q}}$ and $CP_{\mathcal{Q}}$ are solved to optimality. Let π be the vector of dual values associated with constraints (5.5) and y the dual value associated with constraint (5.6).

Lemma 5.2.6. *Let $\bar{c}_j = c_j - \pi^\top A_j$ be the reduced cost of variable x_j. We have $\bar{c}_j \geq z_{\mathcal{Q}}^{CP}$, $j \in \{1, \ldots, n\}$ and $\bar{c}_j = z_{\mathcal{Q}}^{CP}$ for variables such that $v_j > 0$.*

Proof. The dual to $CP_{\mathcal{Q}}$, denoted $DCP_{\mathcal{Q}}$, is

$$
\begin{aligned}
(DCP_{\mathcal{Q}}) \qquad z_{\mathcal{Q}}^{DCP} = \max_{\pi, y} \quad & y && (5.11)\\
\text{s.t.} \quad & c_l - \pi^\top A_l = 0 \quad l \in \mathcal{Q} && (5.12)\\
& c_j - \pi^\top A_j \geq y \quad j \in I_{\mathcal{Q}} && (5.13)\\
& y \in \mathbb{R}, \pi \in \mathbb{R}^m. && (5.14)
\end{aligned}
$$

So, $z_{\mathcal{Q}}^{CP} = z_{\mathcal{Q}}^{DCP}$ and represents the maximum value of y (the minimum reduced cost). □

Proposition 5.2.7. *If the optimal solution to $RP_{\mathcal{Q}}$ is not optimal to $\mathbb{P}$, then $z_{\mathcal{Q}}^{CP} < 0$.*

Proof. Similar results have been proved in Zaghrouti et al. (2014) where $\mathcal{Q} = \mathcal{Q}_s$ and in Elhallaoui et al. (2011) where integrality constraints (5.3) are not considered. The idea of the proof is simple. Suppose that $z_{\mathcal{Q}}^{CP} \geq 0$ and $RP_{\mathcal{Q}}$ is solved to optimality by adding facets like in the proof of Proposition 5.2.5. By Lemma 5.2.6, all variables will have nonnegative reduced costs. That means that the actual solution is optimal to $\mathbb{P}$, which is a contradiction. □

The following proposition provides a lower bound on the optimal objective value of $\mathbb{P}$ using the RP–CP decomposition. Such a lower bound can be found by solving the linear relaxation of the problem, but this can be computationally more expensive. We instead find a good lower bound using the information provided by the RP and CP.

Proposition 5.2.8. *Let $\bar{z}$ be the current optimal objective value of $RP_{\mathcal{Q}}$ and $z^\star$ the value of an optimal solution to $\mathbb{P}$. We have $\bar{z} + m \cdot z_{\mathcal{Q}}^{CP} \leq z^\star \leq \bar{z}$.*

Proof. By Lemma 5.2.6, the reduced costs are lower bounded by $z_{\mathcal{Q}}^{CP}$. The best improvement from entering a variable into the basis with a full step (not larger than 1) is $z_{\mathcal{Q}}^{CP}$, and the maximum number of nonzero variables in an optimal basis of $\mathbb{P}$ is m. Thus, the objective value cannot be improved by more than $m \cdot z_{\mathcal{Q}}^{CP}$. □

We can easily build an example where this bound is reached. Consider m columns having their reduced cost equal to $z_{\mathcal{Q}}^{CP}$ and forming an identity matrix. These columns can simultaneously enter the basis, and the objective value will be $\bar{z} + m \cdot z_{\mathcal{Q}}^{CP}$. This bound can be used as an indicator of the quality of the integer solution of cost $\bar{z}$. This is particularly useful for stopping the solving process when the current solution quality becomes acceptable. A similar bound has been shown good in Bouarab et al. (2014) for a different context (context of column generation for solving the LP). We notice that in practice we generally know a priori the maximum number of columns corresponding to positive variables of the solution (drivers, pilots) which is 10 to 20 times smaller than the number of constraints m. This tightens the bound. Observe that if $z_{\mathcal{Q}}^{CP} = 0$, then $\bar{z} = z^\star$ because $\bar{z} \leq z^\star \leq \bar{z}$. The lower bound varies with $\mathcal{Q}$. The idea is to find a tradeoff for $\mathcal{Q}$ between two extremes $\mathcal{Q}_s$ and $\{1, \ldots, m\}$ such that the lower bound provided by Proposition 5.2.8 is good enough to use as a criterion to stop the solution process and $RP_{\mathcal{Q}}$ is easy to solve.

5.3 Integral simplex using decomposition (ISUD)

Suppose that we have an integer basic solution s to $\mathbb{P}$ with p positive variables. ISUD uses a particular RP–CP decomposition where 1) $\mathcal{Q}$ is nondegenerate, i.e. $\mathcal{Q} = \mathcal{Q}_s$, and 2) we add integrality constraints to the CP: its solution must be integer according to Definition 5.2.2. We thus obtain what we refer to as $CP_{\mathcal{Q}}^I$, a CP with disjoint column requirement; $CP_{\mathcal{Q}}$ is its relaxation.

ISUD starts by solving $RP_{\mathcal{Q}}$. We can solve $RP_{\mathcal{Q}}$ with any commercial MIP solver, but a simpler approach is as follows. Observe that pivoting on any negative-reduced-cost compatible variable improves the objective value of $\mathbb{P}$ and thus yields an integer descent direction because $RP_{\mathcal{Q}}$ is nondegenerate. If we cannot improve the solution of $RP_{\mathcal{Q}}$ with compatible variables, ISUD solves $CP_{\mathcal{Q}}^I$ to get a group of (more than one) entering variables yielding an integer descent direction. The ISUD algorithm can be stated as follows:

Step 1: *Find a good initial heuristic solution s_0 and set $s = s_0$.*

Step 2: *Get an integer descent direction d either by pivoting on a negative-reduced-cost compatible variable of $RP_{\mathcal{Q}}$ or by solving $CP^I_{\mathcal{Q}}$.*

Step 3: *If no descent direction is found then stop: the current solution is optimal. Otherwise, set $s = s + d$, update $\mathcal{Q}$, and go to Step 2.*

The key ISUD findings of (Zaghrouti et al. (2014)) are the following:

- From an integer solution, a pivot on a negative-reduced-cost compatible variable of $RP_{\mathcal{Q}}$ produces an improved integer solution (in Step 2).
- The sequence of pivots of $RP_{\mathcal{Q}}$ on entering variables identified by $CP^I_{\mathcal{Q}}$ provides an improved integer solution. This result is a corollary of Proposition 5.2.3.
- The $RP_{\mathcal{Q}}$ and $CP^I_{\mathcal{Q}}$ improvements are sufficient to achieve optimality in Step 3.
- $CP_{\mathcal{Q}}$ produces disjoint solutions 50% to 80% of the time, without any branching. The normalization constraint (5.6) plays an important role in the efficiency of ISUD: it favors integrality and helps $CP^I_{\mathcal{Q}}$ to find disjoint solutions with relatively few columns. In fact, ISUD focuses on making a series of easy improvements without intensive combinatorial searching, which is generally caused by a difficult branch-and-bound.
- Optimal solutions have been obtained for crew assignment problems with up to 570 000 variables, usually in minutes.

ISUD is a promising approach but it has some limitations. First, if $CP_{\mathcal{Q}}$ fails to find an integer direction, its structure cannot easily be exploited (by commercial solvers) for branching or cutting purposes, because of the normalization constraint. $CP^I_{\mathcal{Q}}$ needs a sophisticated specialized branching scheme (see Zaghrouti et al. (2014), Subsection 3.3 for more details). The tests in Zaghrouti et al. (2014) use a heuristic implementation of the branching scheme: a depth-first branching only where the branch called "0-branch" sets all variables with $v_j > 0$ in the fractional descent direction to 0 and tries to find a completely different descent direction that is integer. There is no guarantee that this implementation will find an optimal integer solution. In fact, ISUD finishes far from the optimal solution up to 40% of the time on difficult instances. Second, the improvement per iteration in the objective value of the RP is rather small because we move from one extreme point to an adjacent one. Third, ISUD cannot directly handle additional linear (non-SPP) constraints because the quasi-integrality property may be lost in the presence of such constraints. There is clearly room for improvement to ISUD.

5.4 Zooming around an improving direction

In this section, we present the zooming approach, an exact approach combining a more flexible RP–CP decomposition and some neighborhood techniques to construct the RP more

efficiently. The new approach aims to resolve the issues raised above about ISUD. Since the optimal RP–CP decomposition is not known a priori, we propose an iterative algorithm called the *zooming algorithm* to find it in a primal paradigm, i.e., while finding a nonincreasing sequence of integer solutions. We start with an initial decomposition that we iteratively update to better approximate an optimal decomposition. The mechanics of the zooming algorithm are discussed in detail in the next two subsections.

5.4.1 Zooming algorithm

We first introduce some basic concepts. Let $\mathcal{w} = \{w_l, l = 1, \ldots, L\}$ be a partition of $\{1, \ldots, m\}$, the index set of the set partitioning constraints, i.e., $\cup_{l\in\{1,\ldots,L\}} w_l = \{1, \ldots, m\}$ such that $w_{l_1} \cap w_{l_2} = \emptyset, \forall l_1 \neq l_2$. In vehicle and crew scheduling applications each set partitioning constraint represents a task (flight, bus trip, ...) to be executed exactly once; these tasks can be ordered by their starting times. A subset w_l can be associated with an artificial column A_{n+l} covering the tasks indexed in it. Let $\mathcal{q} = \{n+1, \ldots, n+L\}$. Columns A_{n+l}, $l \in \{1, \ldots, L\}$ are obviously linearly independent.

Proposition 5.4.1. *If the index set of constraints covered by a column A_j, say T_j, is a union of some of the subsets w_l, also called clusters, then A_j is compatible with $\mathcal{q}$.*

Proof. It suffices to note that a column with a T_j equal to a union of some subsets w_l is in fact a sum, hence a linear combination, of the columns representing these clusters. More formally, there exists $\varepsilon \subset \{1, \ldots, L\}$ such that $T_j = \cup_{l\in\mathcal{E}} w_l$. Consequently, $A_j = \sum_{l\in\mathcal{E}} A_{n+l}$. Thus, this column is compatible with $\mathcal{q}$ according to Definition 5.2.1. □

Remark that $\mathcal{q}$ is uniquely defined by the partition $\mathcal{w}$ and vice versa. In this section, to simplify the presentation we will use the notation $RP_{\mathcal{W}}$ and $CP_{\mathcal{W}}$ instead of $RP_{\mathcal{Q}}$ and $CP_{\mathcal{Q}}$. Actually, $RP_{\mathcal{W}}$ and $CP_{\mathcal{W}}$ are constructed exactly in the same manner described in Section 5.2 but with some subtleties. First, there is only one constraint (anyone) from each w_l, $l \in \{1, \ldots, L\}$ in $RP_{\mathcal{W}}$. The remaining constraints are redundant. The number of constraints of $RP_{\mathcal{W}}$ is then L. Second, the columns of $\mathcal{q}$ are not added to $RP_{\mathcal{W}}$. The cost of the artificial column A_{n+l} is the dual value of the constraint "representing" cluster w_l in $RP_{\mathcal{W}}$. This cost is used to calculate the vector cost of the objective function of $CP_{\mathcal{W}}$.

Recall that $RP_{\mathcal{W}}$ contains columns that are compatible with $\mathcal{w}$ only. To make an incompatible column compatible, we need to update the partition by disaggregating for instance some clusters (breaking them up) in such a way that the condition of Proposition 5.4.1 holds. For example, consider a partition $\mathcal{w}$ with two clusters, $w_1 = \{1, 2, 3\}$ and $w_2 = \{4, 5, 6, 7\}$, and

a column A_{j_0} with $T_{j_0} = \{1, 4, 5\}$. It is clear that T_{j_0} is not a union of w_1 and w_2, so A_{j_0} is incompatible. To make it compatible we can break up the clusters w_1 and w_2: we obtain a new partition $\mathcal{W}'$ with four new clusters, $w'_1 = \{1\}$, $w'_2 = \{2, 3\}$, $w'_3 = \{4, 5\}$, and $w'_4 = \{6, 7\}$. Now T_j is the union of w'_1 and w'_3, so it is compatible with the new partition. There are many techniques to make a column compatible; we could also consider the partition $\mathcal{W}$" with clusters $w''_1 = \{1, 4, 5\}$ and $w''_2 = \{2, 3, 6, 7\}$ so that T_{j_0} is exactly w''_2. The selected technique should benefit from the special structure, if any, of the problem. We use in this paper the one described in Elhallaoui et al. (2010), which is appropriate for vehicle and crew scheduling problems. It exploits the fact that consecutive tasks in vehicle routes are likely to also be consecutive in crew schedules. For instance, if w_1 and w_2 are vehicle routes we would prefer $\mathcal{W}'$ and avoid the second partition $\mathcal{W}''$ because the tasks of $w''_1 = \{1, 4, 5\}$ are not consecutive. The theory presented here is valid for any partitioning technique (i.e., the algorithm validity is independent of the partition but its efficiency is impacted).

We say that a partition is *defined* by an integer solution s if the elements of this partition are defined by the columns (corresponding to nonzero variables) of s, i.e., if $x_l = 1$, then $w_l = T_l$. A direction d is obtained as explained in Section 5.2: $d_j = -\rho\lambda_j$ for $j \in \mathcal{Q}$, $d_j = \rho v_j$ for $j \in I_{\mathcal{Q}}$, and $d_j = 0$ otherwise. The constant ρ is computed such that $s + d$ is an extreme point of LP. The zooming algorithm is as follows:

Step 1: *Find a good heuristic initial solution s_0 and set $s = s_0$, $d = 0$.*

Step 2: *Find a better integer solution in a neighborhood defined by d:*
- *Disaggregate $\mathcal{W}$ according to d.*
- *Construct and solve $RP_{\mathcal{W}}$.*
- *Update s and $\mathcal{W}$: if s is improved, redefine $\mathcal{W}$ according to s.*

Step 3: *Get a descent direction d:*
- *Solve $CP_{\mathcal{W}}$ to get a descent direction d.*
- *If no descent direction can be found or $|z^{CP}_{\mathcal{W}}|$ is small enough then stop: the current solution is optimal or near optimal.*
- *Otherwise, go to Step 2.*

In Step 1, we need an initial integer solution to start up the algorithm; it can be found by a heuristic technique. In some cases, for example when reoptimizing after a perturbation (such as some flights cancellation), the initial solution could simply be the currently implemented solution. Many consecutive tasks in this initial solution are also consecutive in an optimal solution meaning that a good percentage of what we call *primal information* is preserved. The quality of the primal information in the initial solution impacts hugely the performance of the algorithm.

In Step 2, we disaggregate the current partition $\mathcal{W}$ by making the (incompatible) variables of d, i.e., in the CP solution, compatible, as discussed above. We populate the RP with variables (columns) that are compatible with the new partition $\mathcal{W}$. $RP_{\mathcal{W}}$ is a neighborhood of s around d as will be explained in the next subesection. The direction d is used to guide the search to an area of potential improvement. $RP_{\mathcal{W}}$ is of small size and good properties as discussed below. In the general case, it can be solved by any MIP solver, but if d is integer, we do not need a solver. We can easily update the current integer solution s by replacing it by $s+d$. We "reaggregate" the partition by redefining it according to a better integer solution implying fewer clusters. We thus avoid increasing the size of the RP.

In Step 3, we solve the CP to get a descent direction d. If $z^{CP}_{\mathcal{W}} \geq 0$ the current solution is optimal and obviously an optimal decomposition is reached. If $|z^{CP}_{\mathcal{W}}| \leq \frac{\epsilon}{m}$, the current solution is close to optimality according to Proposition 5.2.8. ϵ is a predetermined threshold. The CP is easy to solve as discussed in the next subsection. The next proposition discusses the convergence of the algorithm.

Proposition 5.4.2. *The zooming algorithm provides an optimal solution to $\mathbb{P}$ in a finite number of iterations.*

Proof. The objective value decreases by at least 1 if $RP_{\mathcal{W}}$ improves the current integer solution in Step 2 either by updating it using an integer d or by solving the RP using an MIP solver. If d is fractional, we disaggregate the partition $\mathcal{W}$. In this case, the number of rows of $RP_{\mathcal{W}}$ is increased by at least one. As long as the current integer solution is not optimal to $\mathbb{P}$, $CP_{\mathcal{W}}$ will find a descent direction as claimed by Proposition 5.2.7. Note that while we do not improve the current integer solution and $CP_{\mathcal{W}}$ succeeds in generating descent directions, we disaggregate $\mathcal{W}$. In the worst case, the RP will coincide with the original problem after at most m disaggregations, where m is the number of constraints. □

For example, if we assume that the cost vector $c \geq 0$, then the objective value has a lower bound of 0. The number of (major) iterations has, therfore, an upper bound of $m \times \sum_j c_j$.

5.4.2 Insights into the zooming algorithm

When we disaggregate the partition we ensure that the columns of s are still compatible with the new partition. For example, w_1 and w_2 are compatible with $\mathcal{W}'$ (and incompatible with $\mathcal{W}''$). The incompatible columns of d are of course made compatible with the new partition. We can easily show that s is an integer solution to $RP_{\mathcal{W}}$ and $s+d$ is a basic feasible solution to the linear relaxation of $RP_{\mathcal{W}}$ because it is an extreme point. We can use $s+d$ to provide

the MIP solver of $RP_{\mathcal{W}}$ with an initial feasible basis, thus avoiding a phase I in the simplex algorithm. So, the solution of $CP_{\mathcal{W}}$ is useful even if d is fractional. Also, s can serve as an upper bound. The MIP solver may use s to eliminate some useless branches ($RP_{\mathcal{W}}$ has smaller gap as discussed below) or to find an improved integer solution using heuristics.

Because s and $s + d$ are solutions to the linear relaxation of $RP_{\mathcal{W}}$, we can say that $RP_{\mathcal{W}}$ is actually a neighborhood around the improving direction from s to $s + d$. Observe that at least an improved integer solution is adjacent to s since SPP is quasi-integral, if s is of course not optimal. If the zooming considers a neighborhood large enough, we are likely to find this improved solution in the neighborhood in the early phases of the algorithm as will be explained below. We emphasize that $s + d$ is adjacent to s since d is minimal. Consequently, the variables of the $CP_{\mathcal{W}}$ solution cover a small portion of the constraints, especially in the lower phases. We therefore do not disaggregate the partition too much and $RP_{\mathcal{W}}$ remains tractable. We thus zoom around a small region with a high potential. Figure 5.1 illustrates the neighborhood around an improving direction.

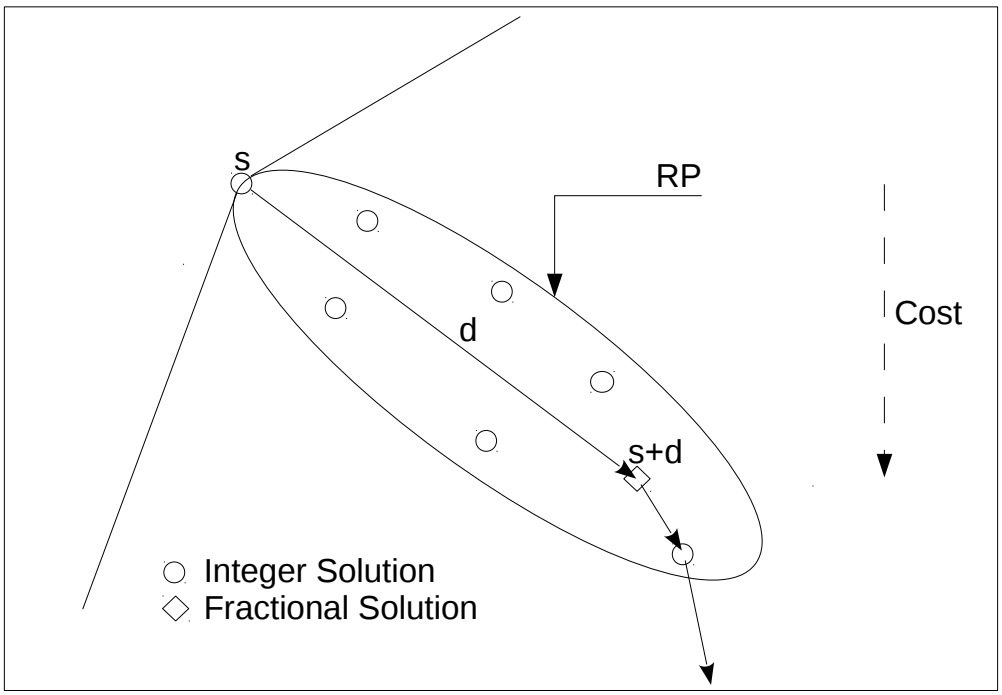

Figure 5.1 Zooming around an improving direction

There are many ways of determining a neighborhood by populating $RP_{\mathcal{W}}$ in Step 2. Populating $RP_{\mathcal{W}}$ is equivalent to a partial pricing of the columns, because we consider only a subset of the columns of A in $RP_{\mathcal{W}}$. We suggest performing the partial pricing based on a function, say $\sigma(j)$, of the reduced cost, *the incompatibility degree*, the number of nonzeros (non zeros

elements), and other relevant attributes of a column A_j. The incompatibility degree can be mathematically defined as the distance from column A_j to the vector subspace generated by the columns of the current integer solution. Of course, if the column is in this vector subspace, i.e., it is a linear combination of the current integer-solution columns; this distance will be 0, indicating that the column is compatible. If the distance (incompatibility degree) is positive, the column is incompatible.

This distance can be computed in various ways, and the computation way plays a key role in the efficiency of the algorithm. It should exploit the specific structure of the problem. We compute this distance, like in Zaghrouti et al. (2014), as the number of additional clusters needed to make a column compatible. In other words, given a partition w and a column A_j, the number of incompatibilities (the incompatibility degree) of A_j with respect to w is given by $k_j^{\mathcal{W}} = \sum_{l=1}^{L} k_{lj}$, where k_{lj} is equal to 1 if column j covers some, but not all, of the tasks in w_l; and 0 otherwise.

A column A_j and its associated variable x_j are said to be *k-incompatible* with respect to a partition w if $\sigma(j) = k$. In this paper, $\sigma(j)$ is the number of incompatibilities of column j. Thus, in our example, the column $A_{j_0} = \{1, 4, 5\}$ is 2-incompatible. A more sophisticated σ that takes into account the reduced cost and other attributes should lead to a better performance. Compatible columns and variables are called *0-incompatible*. We define the phase number as follows:

Definition 5.4.3. *$RP_{\mathcal{W}}$ and $CP_{\mathcal{W}}$ are said to be in phase k when only the variables that are q-incompatible with $q \leq k$ are priced out; k is called the phase number.*

We use a predetermined sequence of phases with increasing σ to accelerate the solution of $RP_{\mathcal{W}}$ in Step 2 and $CP_{\mathcal{W}}$ in Step 3. This generalizes the multiphase strategy introduced in Elhallaoui et al. (2010). Each phase corresponds to a different level of partial pricing. If $RP_{\mathcal{W}}$ or $CP_{\mathcal{W}}$ fails in phase k, we go to the subsequent phase. To ensure the exactness of the algorithm, we end with a phase where all the variables (resp. compatible variables) are priced out for $CP_{\mathcal{W}}$ (resp. $RP_{\mathcal{W}}$). The advantages of the multiphase strategy are discussed below and confirmed by the experiments reported in Section 5.5. In the remainder of this paper, the *current (integrality) gap* refers to the gap between the current integer solution and the LP solution for the problem defined by the context (i.e., RP defined by the context: compatible columns and rows considered).

Proposition 5.4.4. *We have:*

a. The current (integrality) gap of $RP_{\mathcal{W}}$ in phase k is smaller than the current gap of the original problem.

b. The current (integrality) gap of $RP_{\mathcal{W}}$ increases with the phase number.

Proof. a. It is easy to see that $RP_{\mathcal{W}}$ is a restriction of $\mathbb{P}$. Therefore, the current gap of $RP_{\mathcal{W}}$ is smaller than the current gap of $\mathbb{P}$.

b. By Definition 5.4.3, all the k-incompatible columns are present in $RP_{\mathcal{W}}$ in phase $k+1$. Therefore, $RP_{\mathcal{W}}$ in phase k is a restriction of $RP_{\mathcal{W}}$ in phase $k+1$, and it follows that the current gap of $RP_{\mathcal{W}}$ increases with the phase number.

□

This proposition suggests starting with lower phases and increasing the phase number as necessary, because it is much easier to solve the $RP_{\mathcal{W}}$ using an MIP solver when the gap is small. This is consistent with this primal approach that locally improves the current integer solution.

Bouarab et al. (2014) have shown that the number of nonzeros in the constraint matrix of $CP_{\mathcal{W}}$ in phase k is less than $(k+1) \times q_k$, where q_k is the number of columns of $CP_{\mathcal{W}}$. The number of nonzeros in the constraint matrix of $RP_{\mathcal{W}}$ in phase k is also small. The density of $CP_{\mathcal{W}}$ and $RP_{\mathcal{W}}$ depends much more on the disaggregation procedure and the σ function (phase number) than on the density of the original problem. Recall that the number of rows in the RP is L, the number of clusters, because there is only one constraint per cluster in the RP, thus reducing the dimension.

This shows that $RP_{\mathcal{W}}$ and $CP_{\mathcal{W}}$ are easy to solve, especially in the lower phases. In practice we reach optimality after four or five phases (on average) for many vehicle and crew scheduling problems. The number of nonzeros per column would not exceed 4-5 in practice (instead of 40 for some instances of the original problem). For instance, an $RP_{\mathcal{W}}$ in a lower phase with a low density is relatively easy to solve by commercial solvers such as CPLEX. We can use the branching and cutting of such commercial solvers locally and effectively on SPPs that are thirty times or more smaller than the original problem. Commercial solvers are known to be efficient on small to moderate SPPs; they can quickly improve when possible the objective value at each iteration.

5.5 Experimentation

The goal of our tests is to compare the performance of the zooming algorithm (ZOOM) with that of ISUD. We did our tests on a 2.7 Ghz i7-2620M Linux machine. The solver is CPLEX 12.4; it is used to solve the RP as an MIP and the CP as an LP.

5.5.1 Instances

We use the same instances that were used to test ISUD against CPLEX. The instances are in three categories: small (800 rows x 8900 columns), medium (1200 x 139000), or large (1600 x 570000). Small (aircrew scheduling) instances are the largest, in terms of the number of constraints, and the most "difficult" in the OR Library test bed. Medium and large (driver and bus scheduling) instances are difficult. The related problem and parameters are exactly the same described in Haase et al. (2001). In total, we have 90 instances, 30 instances in each category. Note that we do not compare with CPLEX simply because within a time limit of 10 hours, CPLEX cannot find a feasible solution for medium and large instances as reported by Rosat et al. (2016).

The accompanying initial solutions for those instances were produced using a perturbation process as explained in Zaghrouti et al. (2014). The instance difficulty increases as the so-called *unperturbed ratio* decreases. The authors developed a simulation technique to obtain a feasible initial solution with a certain level of good primal information similar to the initial solutions generally available in practice for crew scheduling problems. We measure the primal information by the percentage of consecutive pairs of tasks in the initial solution that remain grouped in the optimal solution. In vehicle and crew scheduling problems, many tasks that are consecutive in the vehicle routes are also consecutive in the crew schedules. We observe the same property when we reoptimize a planned solution after perturbation, for example, after some flights cancellation.

Given an SPP instance with a known optimal integer solution, the perturbation process produces a new instance with the same optimal integer solution and an initial integer solution that is a certain distance from the optimal one. It does this by randomly cutting and recombining the schedules (columns) of the known solution.

The perturbation process particularly simulates the perturbed planned solutions (flights cancellation for instance) and consists in randomly selecting two of the columns in the (planned) solution and replacing them with two different columns covering the same tasks. It chooses two tasks belonging to two different columns in the solution and connects them to create a new column. This process is repeated until the number of unchanged columns is below a certain percentage of the total number of columns in the solution. This parameter is set to 50%, 35%, and 20% for "low", "moderate", and "severe" perturbation. The newly generated columns are added to the problem and given a cost equal to the maximum cost of all the columns in the problem. The perturbed columns are often selected for supplementary perturbations, and the result can be far from the initial solution. Note that the optimal solution columns are not removed from the problem. This method can obtain many instances (with

many starting solutions) from an initial problem; more details can be found in Zaghrouti et al. (2014). The instances and their accompanying initial solutions are available for the OR community.

5.5.2 Numerical results and some implementation tips

The average size of the MIPs solved in Step 2 of ZOOM is small: Table 5.1 compares the average numbers (over 30 instances) of nonzero elements (total (NZs) and per column (NZs/col.)) of the original problem (Orig. Pb.) and the RPs (MIP) for the small, medium, and large instances. The number of nonzeros in the RP (MIP) is reduced by huge factors varying from 30 to more than 100 on average. The density is also reduced by a factor of 2. Therefore, we do not use the multiphase strategy when solving the RP in Step 2 of ZOOM; it is used only for CP. Recall that the distance of a column to a given solution is its incompatibility degree, as explained in Section 5.4. We consider first the columns that are not too far from the best solution found and we increase this distance as needed. More precisely, we start by phase 1 when we solve $CP_{\mathcal{W}}$ in Step 3 of ZOOM. As long as we do not find a descent direction, we increase by 1 the phase number, i.e., we go to the next phase and solve $CP_{\mathcal{W}}$ again. The maximum phase number (maximum degree of incompatibility), used for the multiphase strategy, is set to 8. This value is reached in only one instance. We use the same stopping criterion (the best) for ISUD as in Zaghrouti et al. (2014). We would like to mention that the results of ISUD are improved because the version of CPLEX has changed from 12.0 used in Zaghrouti et al. (2014) to 12.4 here.

Table 5.1 ISUD vs. ZOOM (nonzeros)

Instance size	Orig. Pb.		MIP	
	NZs	NZs/Col.	NZs	NZs/Col.
Small	74000	8	720	3
Medium	2732000	21	22830	12
Large	10942000	19	314232	11

The first set of tables (Tables 5.2, 5.3, and 5.4) show from left to right: the unperturbed ratio for the instance (Orig. Col. %), the identifier for the instance (Instance #), the distance of the initial solution from the optimal one as a percentage of the optimal objective value (Instance Err. %), the distance to optimality of the solution found beside the total computational time in seconds (Objective Err. % beside Objective Time) for both ISUD and ZOOM, the total

number of MIPs built and solved by ZOOM (Num), the number of MIPs that improved the current integer solution (Num+), the average number of rows in MIPs (Rows), the average number of columns in MIPs (Cols), the average number of nonzero elements in MIPs (NZs), and the average computational time in seconds for MIPs (Time).

The error percentages (Err. %) may be greater than 100 because they compare the objective value of a solution to the optimal objective value ((solution value - optimal value) / optimal value) $\times 100$). The MIP values (rows, columns, NZs, and time) are average values aggregated over all the MIPs solved by ZOOM. When no MIP is called, no information is available.

The tables in the second set (Tables 5.5, 5.6, and 5.7) compare the CPs for ISUD and ZOOM. The first two columns show the unperturbed ratio (Orig. Col. %) and the identifier for the instance (Instance #). Then, for both ISUD and ZOOM, the rest of columns show: the highest phase in which an integer solution is found (Phase), the total number of solutions, i.e. directions, found by CP (Total) that is also the number of iterations, the number of disjoint solutions (Disj.), the maximum size of the disjoint solutions (Max.) and the average size of the disjoint solutions (Avg.).

In both sets of tables, we have included average lines in bold to compare the average behavior of the two algorithms.

The results show that ZOOM outperforms ISUD in terms of solution quality, and it is generally slightly faster. Interestingly, ZOOM obtains solutions that are very close to optimality or optimal in all cases. We observe that ZOOM has the smallest error ratio average (almost 0% on average and a peak of 0.8% for ZOOM against 7.28% on average and a peak of 200% for ISUD). ZOOM is better than ISUD for all instances except for some small instances where ISUD is slightly and insignificantly better.

When we solve RP in Step 2 of ZOOM, we consider all columns that are compatible with w, including those that are in phases higher than the actual phase of CP. To reach these columns, ISUD (its CP) has to go to similar high phases which complicates its execution: too much time for solving CP (too many columns to consider) and the directions found by CP are too much fractional because as we observed **non-integrality increases with the incompatibility degree, i.e., with the phase number**. In a highly fractional direction, the number of variables (v_j) taking nonzero values is huge actually. Consequently, ISUD stops with a poor solution due to its depth-first branching (setting to 0 these variables) as explained in Section 5.3. This is avoided in ZOOM.

Approximately 52% of the MIPs improve the current solution for difficult instances (see the sets with the unperturbed ratio equal to 20% in Tables 5.2-5.4). When the MIP does not

improve the current solution, we use this information as an indicator to increase the phase number and often we find an integer direction the next iteration. For some easy instances for which CP does not encounter a fractional solution, ZOOM does not build any MIPs and consequently it behaves the same way as ISUD.

ZOOM is on average slightly faster than ISUD, about 5% faster on average on difficult instances because ISUD is faster in some cases when it stops prematurely with a poor solution quality on 35% of the small and medium instances that are difficult (i.e., set with unperturbed ratio equal to 20%). Solving the MIPs is fast as expected (see discussion of Section 5.4): less than 3 seconds, on average, for the small and medium instances and around 9 seconds for the large instances. The number of columns considered is 20 to 30 times smaller than the total number of columns.

The results in Tables 5.5–5.7 show that ZOOM explores less phase numbers than ISUD because some useful columns in higher phases are considered earlier in RP as mentioned above. Also, ZOOM solves fewer CPs and consequently does few iterations (8% less than ISUD on difficult instances). This can be explained by the fact that ZOOM uses also RP as an MIP to improve the current solution, interestingly by longer steps (ZOOM explores, in contrast to ISUD, some non adjacent solutions). For the same reason, the maximum and average sizes of the CP solutions are also smaller in ZOOM.

The lower bounds given by Proposition 5.2.8 are very good. For the most difficult (with regard to the phase in which the best solution is obtained, i.e. phase 8) instance (#87), $z_{\mathcal{W}}^{CP} = 0$ when we consider only columns with incompatibility degree not larger than 8 and $z_{\mathcal{W}}^{CP} = -7$ when we consider all columns. The cost of the (optimal) solution found is 3 302 462. From all above, we clearly see that the numerical results confirm the theoretical results presented in Sections 5.2 and 5.4.

Table 5.2 ISUD vs. ZOOM (small instances)

			ISUD		ZOOM							
Orig.	Instance		Objective		Objective		MIP Averages					
Col. %	#	Err. %	Err. %	Time	Err. %	Time	Num.	Num+	Rows	Cols	NZs	Time
50%	1	360.6	0	7	0	7	4	0	127	198	667	0
	2	357.4	0	6	0	5	4	0	127	198	669	0
	3	347.7	0	6	0	5	4	0	127	201	684	0
	4	356.8	0.08	7	0	6	5	1	123	183	590	0
	5	353.3	0	6	0	5	4	0	127	200	676	0
	6	352.8	0	5	0	5	4	0	127	202	682	0
	7	364.2	0	6	0	5	4	0	127	198	669	0
	8	368	0	7	0.03	6	4	0	128	199	667	0
	9	360.1	0	6	0	5	4	1	130	217	751	0
	10	366.9	0	7	0	6	3	0	132	213	749	0
		358.78	***0.01***	**6.3**	**0**	**5.5**	**4**	**0.2**	**127.5**	**200.9**	**680.4**	**0**
35%	11	472.1	0.03	6	0	5	4	1	128	196	664	0
	12	472.8	0	8	0.09	9	5	2	129	192	633	0.2
	13	464.7	0	5	0	5	4	0	127	200	675	0
	14	468.4	0	6	0	6	5	1	140	265	961	0
	15	463.5	0	6	0	5	3	0	132	214	753	0
	16	466.3	0	8	0.06	6	6	2	127	180	566	0
	17	456.3	0	11	0.8	11	7	4	42	139	563	0.01
	18	471.6	0	5	0	5	4	0	127	200	676	0
	19	469.2	0.64	8	0	9	4	1	125	189	626	0
	20	464.6	0	6	0	6	6	2	122	175	552	0
		466.95	***0.07***	**6.9**	**0.1**	**6.7**	**4.8**	**1.3**	**119.9**	**195**	**666.9**	**0.02**
20%	21	570.2	0	8	0	8	2	0	140	248	933	0
	22	561	0	12	0	11	5	2	173	438	1778	0.2
	23	559.9	138.19	9	0.05	9	5	2	120	165	519	0
	24	557.7	0	9	0	9	3	0	132	214	754	0
	25	562.5	0	12	0.25	14	9	6	128	212	692	0
	26	561	0	9	0	9	4	1	127	192	646	0
	27	573.1	0	8	0.05	8	4	0	126	191	637	0
	28	569.6	0.02	10	0	9	5	3	126	185	614	0
	29	569.3	0	12	0	10	4	0	126	188	624	0
	30	573.7	200.63	8	0.09	13	13	10	126	199	615	0
		565.8	***33.88***	**9.7**	**0.04**	**10**	**5.4**	**2.4**	**132.4**	**223.2**	**781.2**	**0.02**

Table 5.3 ISUD vs. ZOOM (medium instances)

			ISUD		ZOOM							
Orig.	Instance		Objective		Objective		MIP Averages					
Col. %	#	Err. %	Err. %	Time	Err. %	Time	Num.	Num+	Rows	Cols	NZs	Time
50%	31	53.43	0	40	0	41	1	0	100	153	1021	0
	32	51.39	0	57	0	58	1	0	100	153	1021	0
	33	53.43	0	56	0	55	1	0	100	153	1021	0
	34	53.44	0	55	0	54	1	0	100	153	1021	0
	35	51.38	0	91	0	85	3	1	207	1103	13192	0
	36	51.38	0	41	0	41	1	0	100	153	1021	0
	37	51.38	0	104	0	97	2	0	239	1367	16629	0
	38	51.38	0	98	0	93	2	0	239	1370	16664	0
	39	51.38	0	124	0	104	3	0	297	2084	26049	0
	40	51.38	0	39	0	38	1	0	100	153	1021	0
		52.00	**0**	**70.5**	**0**	**66.6**	**1.6**	**0.1**	**158.2**	**684.2**	**7866**	**0**
35%	41	65.77	0	136	0	127	3	1	189	942	11189	0
	42	65.77	0	63	0	63	1	0	100	153	1021	0
	43	65.77	10.27	92	0	101	5	2	196	1043	12418	0.2
	44	67.83	12.33	130	0	139	4	2	363	4667	64402	2.75
	45	65.77	0	180	0	147	6	4	207	2067	28428	1.78
	46	67.83	0	105	0	101	2	0	238	1361	16443	0
	47	67.82	0	193	0	125	3	1	310	3248	41342	0.67
	48	65.77	0	75	0	70	2	0	239	1369	16645	0
	49	65.77	0	107	0	90	3	1	211	1094	13016	0
	50	67.83	0	114	0	98	2	0	239	1370	16656	0.5
		66.59	**2.26**	**119.5**	**0**	**106.1**	**3.1**	**1.1**	**229.2**	**1731.4**	**22156**	**0.59**
20%	51	82.21	0	213	0	109	3	1	439	5162	65688	1
	52	82.21	0	146	0	141	2	0	238	1360	16445	0.5
	53	82.21	0	78	0	77	1	0	100	153	1021	0
	54	82.21	0	133	0	123	2	0	239	1371	16658	0
	55	82.21	41.11	75	0	106	2	1	497	9228	120243	6
	56	82.21	63.72	38	0	86	2	1	552	13869	194241	13
	57	82.21	0	222	0	149	3	1	370	3110	40936	0.67
	58	82.21	22.61	111	0	170	4	1	359	4660	64203	3.25
	59	82.21	0	101	0	95	2	0	239	1369	16641	0
	60	82.21	32.88	63	0	81	2	1	455	7007	94194	2
		82.21	**16.03**	**118.0**	**0**	**113.7**	**2.3**	**0.6**	**348.8**	**4728.9**	**63027**	**2.64**

Table 5.4 ISUD vs. ZOOM (large instances)

			ISUD		ZOOM							
Orig.	Instance		Objective		Objective		MIP Averages					
Col. %	#	Err. %	Err. %	Time	Err. %	Time	Num.	Num+	Rows	Cols	NZs	Time
50%	61	51.88	0	652	0	655	0	0				
	62	50.34	0	919	0	912	0	0				
	63	50.34	0	307	0	307	0	0				
	64	50.35	10.68	789	0	916	1	1	716	35387	377443	9
	65	50.34	0	1355	0	1364	0	0				
	66	51.87	0	274	0	273	0	0				
	67	50.34	10.68	453	0	1141	1	1	681	27519	304478	7
	68	50.35	0	623	0	628	0	0				
	69	51.87	0	587	0	587	0	0				
	70	50.34	0	1338	0	1342	0	0				
		50.80	**2.14**	**729.7**	**0**	**812.5**	**0.2**	**0.2**	**698.5**	**31453**	**340960**	**8**
35%	71	65.60	0	804	0	797	0	0				
	72	65.60	0	695	0	695	0	0				
	73	65.59	0	617	0	611	0	0				
	74	65.60	0	1266	0	1257	0	0				
	75	65.60	15.26	352	0	664	1	1	727	30366	340469	8
	76	65.60	0	1508	0	1501	0	0				
	77	65.60	19.83	251	0	350	1	1	755	31874	352756	10
	78	65.60	0	1929	0	1938	0	0				
	79	65.59	0	1578	0	1447	1	1	673	22763	240986	6
	80	65.60	10.68	416	0	681	1	1	651	25743	272085	7
		65.60	**4.58**	**941.6**	**0**	**994.4**	**0.4**	**0.4**	**701.5**	**27686.5**	**301574**	**7.75**
20%	81	82.38	0	2126	0	2128	0	0				
	82	80.85	0	1498	0	1219	1	1	761	31931	347784	8
	83	80.85	0	1473	0	1475	0	0				
	84	80.86	0	1695	0	1393	1	1	717	35626	399536	9
	85	80.86	0	1719	0	1578	1	1	686	29838	317317	7
	86	80.86	0	1031	0	1010	0	0				
	87	80.85	0	1675	0	1390	2	1	644	21032	222389	19
	88	80.85	0	2018	0	2010	0	0				
	89	80.86	0	1299	0	1174	1	1	741	40274	453855	11
	90	80.86	0	1666	0	881	1	1	596	15818	164729	4
		81.01	**0**	**1620**	**0**	**1425.8**	**0.7**	**0.6**	**690.83**	**29086.5**	**317601.7**	**9.67**

Table 5.5 ISUD CP vs. ZOOM CP (small instances)

		ISUD					ZOOM				
Orig.		CP Solutions					CP Solutions				
Col. %	#	Phase	Total	Disj.	Max.	Avg.	Phase	Total	Disj.	Max.	Avg.
50%	1	4	35	29	6	2.4	4	33	29	6	2.4
	2	3	38	31	7	2.4	3	36	31	7	2.4
	3	4	33	27	4	2.4	4	31	27	4	2.4
	4	5	56	39	8	2.9	3	42	36	7	2.7
	5	4	33	27	4	2.4	4	31	27	4	2.4
	6	4	33	27	5	2.6	4	31	27	5	2.6
	7	4	38	32	5	2.2	4	36	32	5	2.2
	8	4	41	33	9	2.9	4	36	31	9	2.8
	9	5	32	26	7	2.5	4	30	25	6	2.5
	10	4	36	31	10	2.5	4	34	31	10	2.5
		4.1	**37.5**	**30.2**	**6.5**	**2.52**	**3.8**	**34**	**29.6**	**6.3**	**2.49**
35%	11	5	42	33	7	3	4	35	31	11	3
	12	6	39	29	19	3.4	4	40	34	8	3
	13	3	37	31	7	2.6	3	35	31	7	2.6
	14	5	35	27	17	3.1	4	34	29	6	2.6
	15	4	37	32	5	2.8	4	35	32	5	2.8
	16	5	50	39	8	2.8	4	39	33	7	2.6
	17	5	43	36	10	3.5	5	45	37	7	3.5
	18	3	46	39	5	2.3	3	44	39	5	2.3
	19	4	46	37	10	3.1	4	42	38	10	3.1
	20	5	40	30	7	2.7	6	22	20	3	2.2
		4.5	**41.5**	**33.3**	**9.5**	**2.93**	**4.1**	**37.1**	**32.4**	**6.9**	**2.77**
20%	21	5	44	40	15	3.3	5	42	40	15	3.3
	22	5	53	44	8	3.1	5	46	41	8	3.2
	23	5	44	23	18	4.3	5	31	26	14	4.5
	24	4	48	43	13	3.3	4	46	43	13	3.3
	25	5	52	43	13	3.6	5	52	42	9	3.2
	26	5	42	33	12	3.4	4	36	32	12	3.4
	27	4	47	40	8	2.8	4	43	39	8	2.8
	28	6	51	40	8	3.2	5	45	40	7	3
	29	4	63	51	14	3.3	4	47	41	14	3.4
	30	4	43	31	11	3.5	5	47	34	17	3.8
		4.7	**48.7**	**38.8**	**12**	**3.38**	**4.6**	**43.5**	**37.8**	**11.7**	**3.39**

Table 5.6 ISUD CP vs. ZOOM CP (medium instances)

		ISUD					ZOOM				
Orig.		CP Solutions					CP Solutions				
Col. %	#	Phase	Total	Disj.	Max.	Avg.	Phase	Total	Disj.	Max.	Avg.
50%	31	4	15	14	3	2.1	4	15	14	3	2.1
	32	5	17	16	3	2.1	5	17	16	3	2.1
	33	5	16	15	3	2.2	5	16	15	3	2.2
	34	5	15	14	4	2.4	5	15	14	4	2.4
	35	6	15	12	6	2.4	6	15	12	3	2.2
	36	5	16	15	3	2.1	5	16	15	3	2.1
	37	6	18	16	3	2.1	6	18	16	3	2.1
	38	6	18	16	3	2.1	6	18	16	3	2.1
	39	7	19	15	5	2.3	7	18	15	5	2.3
	40	4	15	14	3	2.1	4	15	14	3	2.1
		5.3	**16.4**	**14.7**	**3.6**	**2.19**	**5.3**	**16.3**	**14.7**	**3.3**	**2.17**
35%	41	0 6	24	21	4	2.3	6	23	20	3	2.2
	42	5	18	17	4	2.3	5	18	17	4	2.3
	43	6	19	15	3	2.3	6	20	15	3	2.3
	44	7	22	17	3	2.2	7	20	16	3	2.2
	45	7	23	19	5	2.5	6	21	15	5	2.5
	46	6	22	20	6	2.3	6	22	20	6	2.3
	47	7	24	20	4	2.4	6	20	17	4	2.4
	48	6	16	14	8	2.7	6	16	14	8	2.7
	49	6	22	19	5	2.3	6	20	17	5	2.4
	50	6	22	20	3	2.2	6	22	20	3	2.2
		6.2	**21.2**	**18.2**	**4.5**	**2.35**	**6**	**20.2**	**17.1**	**4.4**	**2.35**
20%	51	8	22	17	8	3.2	7	16	13	5	2.5
	52	6	26	24	4	2.6	6	26	24	4	2.6
	53	5	25	24	5	2.5	5	25	24	5	2.5
	54	6	25	23	5	2.7	6	25	23	5	2.7
	55	6	16	15	4	2.5	6	17	15	4	2.5
	56	6	9	8	3	2.1	6	10	8	3	2.1
	57	8	27	21	5	2.6	7	23	20	5	2.5
	58	6	21	18	4	2.6	7	23	19	4	2.5
	59	6	23	21	5	2.6	6	23	21	5	2.6
	60	6	16	15	5	2.4	6	17	15	5	2.4
		6.3	**21.0**	**18.6**	**4.8**	**2.58**	**6.2**	**20.5**	**18.2**	**4.5**	**2.49**

5.6 Possible extensions

The zooming approach is globally a primal exact approach. At each iteration, it improves the current integer solution by zooming around an improving fractional direction. In practice, it

Table 5.7 ISUD CP vs. ZOOM CP (large instances)

		ISUD					ZOOM				
Orig.		CP Solutions					CP Solutions				
Col. %	#	Phase	Total	Disj.	Max.	Avg.	Phase	Total	Disj.	Max.	Avg.
50%	61	6	18	18	4	2.4	6	18	18	4	2.4
	62	6	19	19	4	2.2	6	19	19	4	2.2
	63	5	17	17	4	2.2	5	17	17	4	2.2
	64	8	23	16	3	2.1	5	16	15	2	2
	65	7	16	16	4	2.5	7	16	16	4	2.5
	66	5	19	19	3	2.1	5	19	19	3	2.1
	67	7	16	13	4	2.3	7	14	13	4	2.3
	68	6	17	17	4	2.2	6	17	17	4	2.2
	69	6	16	16	4	2.4	6	16	16	4	2.4
	70	7	19	19	3	2.2	7	19	19	3	2.2
		6.3	**18**	**17.1**	**3.7**	**2.25**	**6**	**17.1**	**16.9**	**3.6**	**2.24**
35%	71	6	25	25	3	2.2	6	25	25	3	2.2
	72	6	23	23	5	2.5	6	23	23	5	2.5
	73	6	24	24	5	2.3	6	24	24	5	2.3
	74	7	26	26	5	2.3	7	26	26	5	2.4
	75	6	23	21	5	2.3	6	22	21	5	2.3
	76	7	24	24	5	2.4	7	24	24	5	2.4
	77	5	18	17	5	2.5	5	18	17	5	2.5
	78	5	24	24	5	2.3	7	26	26	5	2.3
	79	8	27	25	4	2.6	7	24	23	4	2.6
	80	6	23	22	7	2.5	6	23	22	7	2.6
		6.2	**23.7**	**23.1**	**4.9**	**2.39**	**6.3**	**23.5**	**23.1**	**4.9**	**2.4**
20%	81	7	27	27	7	3.4	7	27	27	7	3.4
	82	8	24	22	13	3.7	7	19	18	6	3
	83	7	31	30	7	2.5	7	31	30	7	2.5
	84	8	23	21	13	3.4	7	18	17	4	2.4
	85	8	30	27	8	3.3	7	26	25	8	3.2
	86	6	32	32	6	2.6	6	32	32	6	2.6
	87	9	25	21	22	3.8	8	20	18	22	3.7
	88	7	31	31	8	3.2	7	31	31	8	3.2
	89	8	21	19	10	3.6	7	17	16	6	2.9
	90	7	34	33	4	2.6	6	32	31	4	2.6
		7.5	**27.8**	**26.3**	**9.8**	**3.21**	**6.9**	**25.3**	**24.5**	**7.8**	**2.96**

improves on the ISUD algorithm of Zaghrouti et al. (2014). Possible extensions of this work include:

— Considering non-SPP constraints: non-SPP constraints are added to many SPPs to model various regulations. For example, in aircrew scheduling there may be a con-

straint limiting the number of flying hours per base. These constraints represent less than 10% of the total number of constraints, and the problem becomes an SPP with side constraints. The zooming algorithm can easily handle these additional constraints by putting them in the RP. Other options could be studied.

— Accelerating the solution process by obtaining several orthogonal descent directions, good in practice for large problems, by successively solving appropriate CPs. To do this, we solve the CP, remove the variables with positive values in the solution and the constraints that they cover, and start again. This gives a multidirectional zoom on a targeted neighborhood around the improving directions. The number of directions should be customizable and adjusted experimentally; in this paper we consider just one direction.

— Obtaining good improving directions by adjusting the coefficients of the normalization constraint or by cutting bad directions: Rosat et al. (2014, 2016) developed the concepts of penalizing and cutting fractional directions. These concepts could be adapted for the zooming approach.

— Designing more sophisticated σ functions to target good neighborhoods around the improving directions: We could integrate developments from metaheuristics concerning neighborhood structures.

— Integrating with column generation: in practice, large SPPs are often solved by column generation. We could explore how to use this column generation technique in the vector space decomposition framework.

CHAPITRE 6 ARTICLE 3 : IMPROVED INTEGRAL SIMPLEX USING DECOMPOSITION FOR THE SET PARTITIONING PROBLEM

Improved Integral Simplex Using Decomposition for the Set Partitioning Problem

Abdelouahab Zaghrouti, Issmail El Hallaoui, François Soumis

Article soumis à Euro Journal on Computational Optimization

6.1 Introduction

The Integral Simplex paradigm was a big step in the research studying the well-known Set Partitioning Problem (SPP). The Integral Simplex reaches an optimal integer solution by finding a sequence of basic integer solutions. It was first introduced by Balas & Padberg (1975) in the early 70s. Balas and Padberg and several other authors presented algorithms for the Integral Simplex: Haus et al. (2001), Thompson (2002), Saxena (2003), and Rönnberg & Larsson (2009). All these algorithms were mainly based on enumerating degenerate bases for which the number is huge. So, the intrinsic degenerate nature of the SPP model was a big obstacle against a useful algorithm for this paradigm in practice. Lately, Zaghrouti et al. (2014) proposed an algorithm that is considered successful in solving SPPs for vehicle and crew scheduling problems (bus driver scheduling and airline crew pairing). The algorithm, called ISUD (Integral Simplex Using Decomposition), is based on the decomposition of the problem into two sub-problems: A first, generally small, sub-problem called Reduced Problem (RP) is simply an SPP restricted to a subset of variables, that are *compatible* with the current integer solution as will be explained in the next section. The second sub-problem, called Complementary Problem (CP), is a descent direction finding problem (on a cone) that includes the remaining variables. CP introduces an extra constraint called the normalization constraint to scale descent directions and ensure boundedness.

ISUD inherits its success, against degeneracy in SPP, from a previous algorithm called IPS (Improved Primal Simplex) introduced by Elhallaoui et al. (2011). IPS increases the efficiency of the primal simplex method when solving degenerate linear programs. ISUD is able to solve, to optimality or near optimality, large-scale vehicle and crew scheduling problems without having to do any exhaustive branching. With the help of a column pricing technique, called multi-phase strategy, the algorithm presents very good performance results.

However, some limitations still prevent ISUD from reaching higher levels of quality of solution and performance. In fact, ISUD presents cases when it fails in moving further towards an optimal integer solution. Recently, three methods addressed this issue for ISUD. Rosat et al. (2014) added the use of cutting planes to ISUD to minimize its need to branching; Zaghrouti et al. (2013) approached this issue by dynamically adjusting the decomposition and concentrating the search within an enlarged neighborhood around a descent direction; and Rosat et al. (2016) tested different weights in the so-called normalization constraint to improve the integrality of the next solution.

The two first methods, mentioned above, could be considered "corrective" in the sense that they try to find alternate ways to make ISUD succeed after it fails. The third method could

be considered "preventive" because it adjusts the CP model to improve its success ratio in finding improved integer solutions.

In this paper, we introduce a "preventive" approach that highly improves the success ratio of ISUD and minimizes the need of corrective actions. The new algorithm, actually, solves a sub-problem (CP) based on an improved model. The new model presents higher chances in directly finding the next better integer solution. When it fails doing so, it also presents an easy extra chance of deducing the next better integer solution without going through a complete branching (more details are given in Section 6.3).

Our improved version of ISUD presents a better overall performance. It reaches better integer solutions faster:

— the quality of reached solutions is improved by using a modified version of the algorithm that is more efficient. The new algorithm reaches optimal solutions without the need of any pricing strategy (considering all the columns of the problem).

— the performance is improved by reducing the number of iterations needed by the algorithm to reach an optimal solution.

This paper is organized as follows: in the next section, we recall a brief description of ISUD; in Section 6.3, we introduce the new model and algorithm and discuss some differences between the standard and the improved versions of ISUD; we present results from our experimentation in Section 6.4; and we end the paper by a conclusion in Section 6.5.

6.2 ISUD

Consider the Set Partitioning model:

$$\min \quad c \cdot x \tag{6.1}$$

$$(P) \quad Ax = e \tag{6.2}$$

$$x_j \quad \text{binary}, \quad j \in N, \tag{6.3}$$

where A is an $m \times n$ matrix with binary coefficients, c an arbitrary vector with non negative integer components of dimension n, $e = (1, \ldots, 1)$ a vector of dimension m and $N = \{1, \ldots, n\}$ the set of indices of all variables in P. Let LP be the continuous relaxation of P.

First, let us introduce some definitions and notation. A variable x_j is said to be compatible with the current integer solution if the corresponding column A_j is a linear combination of some columns corresponding to non null variables of the current integer solution; otherwise it is said to be incompatible. To alleviate statements like the previous one, the term variable and

column are used in an interchangeable manner in this paper. Also, for simplicity hereinafter, the term solution designates both the solution vector and its support (the set of non null variables). The term solution could also designate a reduced solution (a sub-vector of the solution that identifies the solution: the rest of the solution can be deduced from the problem). The same goes for the term descent direction, which is, hereinafter, simply called direction; e.g. the set of entering variables of a descent direction is also referred to as a direction. Let v be a vector and F a set of indices. v_F designates a sub-vector of v composed of components indexed in F. If k is a constant, k_F designates a vector of dimension $|F|$ where each component is equal to k.

The incompatibility degree of a column towards a given integer solution is a measure that represents a distance of the column from the solution. This measure is, in general, set to 0 for compatible columns and to a positive value for incompatible ones. The multi-phase strategy is a partial pricing where only columns having an incompatibility degree less than a certain maximum value are priced out in a given phase. The maximum value defines the phase of the pricing.

The Reduced Problem (RP) is simply a set partitioning problem including only compatible columns (columns from the current integer solution included). The Complementary Problem (CP) is a linear problem containing only incompatible columns and is formulated as in the generalized form studied in Rosat et al. (2016). The formulation is as follows:

$$\min \quad \sum_{j \in I} c_j y_j - \sum_{l \in L} c_l \lambda_l \tag{6.4}$$

$$(CP) \quad \sum_{j \in I} y_j A_j - \sum_{l \in L} \lambda_l A_l = 0 \tag{6.5}$$

$$\sum_{j \in I} \omega_j y_j + \sum_{l \in L} \omega_l \lambda_l = 1 \tag{6.6}$$

$$y_j \geq 0, \lambda_l \geq 0 \tag{6.7}$$

where I is the set of indices of incompatible variables and L the set of indices of columns of the current integer solution. Positive y_j enter the solution and positive λ_l exit the solution. So, CP finds two sets of variables: entering and exiting. The entering set needs to have a negative reduced cost (6.4). Constraints (6.5) are the compatibility constraints, i.e., the combination of entering columns is compatible (a linear combination of some columns of the solution as defined above). Constraint (6.6) is the normalization constraint where ω is the normalization-weight vector (all of its components are non negative). Updating the integer solution is done by a simple exchange between the entering and the exiting sets. In fact, this exchange operation is equivalent to applying a sequence of regular simplex pivots on entering

variables. Here is a summarized description of ISUD:

Step 0: Start with an initial integer solution.

Step 1: Improve the current integer solution using RP.

Step 2: Improve the current integer solution using CP; making some branching if necessary.

Control step: If Step 2 improves the solution, go to Step 1. Otherwise, the current solution is optimal.

Integral Simplex, as a primal approach, should only jump from an integer solution to an improved one. So, ISUD only applies sets of pivots that keep the updated solution in an integral state. Sets of pivots leading to a non integer solution are rejected. An accepted direction found by CP leads to a different integer solution having a better cost. We call it an *integer direction*. The direction is integer *iff* the entering columns are row-disjoint. Otherwise, the current solution and its cost stay the same; we call the rejected direction a *fractional direction*. The entering sets of columns found by CP are compatible with the current integer solution and are, also, non decomposable in the sense that they have no compatible subset (see Proposition 3 from Zaghrouti et al. (2014)). Directions found by CP lead to improved solutions that are adjacent to the current integer solution.

Denote by x^0 a current integer solution to P. The following is an equivalent non linear formulation for CP (see formulation 3 of the *maximum normalized augmentation* problem from Rosat et al. (2016)):

$$(CP) \quad \Leftrightarrow \quad \min \quad \frac{c \cdot d}{\|d\|_\omega} \tag{6.8}$$

$$Ad = 0 \tag{6.9}$$

where $c \cdot d =$ the cost for the direction $d =$ the reduced cost of $x = x^0 + d$ (x is solution to LP). $\|d\|_\omega = \sum_{j \in I} \omega_j |d_j| + \sum_{l \in L} \omega_l |d_l|$ is *the weighted* l_1*-norm* of d.

Proposition 6.2.1. *We have* $\|d\|_\omega = \frac{1}{\lambda_{max}}$ *with* $\lambda_{max} = \max_{l \in L} \lambda_l$

Proof. Let $d^n = \begin{pmatrix} y_I \\ -\lambda_L \end{pmatrix}$ be a solution to CP (the zero part of d^n corresponding to the compatible variables is omitted for simplicity). Consider $x^1 = x^0 + d = x^0 + td^n$ the solution of LP which d leads to. $x^1 = \begin{pmatrix} 0_I + ty_I \\ 1_L - t\lambda_L \end{pmatrix}$. So, the feasibility of x^1 means: $ty_j \leq 1$, $(j \in I)$ and $t\lambda_l \leq 1$, $(l \in L)$. λ_l $(l \in L)$ being larger than their corresponding y_j $(j \in I)$ (i.e. covering some constraint covered by λ_l). The largest t possible is $\frac{1}{\lambda_{max}}$ that is also $\|d\|_\omega$. So, d^n is the

normalized form of d and $\|d\|_\omega$ is the largest step to take along d^n while staying feasible for LP. □

The normalization-weight vector coefficients ω_j ($j \in I$) and ω_l ($l \in L$) are important parameters for the algorithm. For the standard algorithm introduced in Zaghrouti et al. (2014), those coefficients were set to $\omega_j = 1$ ($j \in I$) and $\omega_l = 0$ ($l \in L$). Rosat et al. (2016) tested and compared other variants (see Sub-section 6.3.2 for a summarized description).

6.3 I²SUD, an Improved ISUD

The numerical results for ISUD from Zaghrouti et al. (2014) show that about 10% to 30% of the directions returned by CP are fractional. When this happens, ISUD then tries to correct the situation by applying a simple depth-first branching; if the branching is not quickly successful, it goes to the next phase of the multi-phase strategy to change the subset of columns in CP. By avoiding a complete branching, ISUD escapes from a potential performance bottleneck. It hopes the multi-phase strategy will repopulate CP and lead to an integer direction; which happens quite often. With this smart way of dealing with fractional directions, ISUD reaches optimality for about 80% of the instances from its tests. For the remaining 20%, ISUD stops within a distance from optimality because only fractional directions start to show up in the higher phases.

The power of ISUD is that it favors the conditions preventing fractional directions from appearing during the solving process. When those preventing conditions fail, fractional directions are encountered. Then, some corrective actions like branching or cutting are needed to force the finding of an integer direction.

Our contribution, in this paper, is to introduce I²SUD, a new version of ISUD, that improves, instead, the aspect that makes the power of ISUD: **reinforce** the conditions preventing fractional directions from appearing. We aim to find fractional solutions less often than ISUD (or equivalently find integer solutions more often than ISUD). For this, we introduce an improved model for CP.

6.3.1 Improved model

The new CP ($CP2$) is a modified version of its generalized form. $CP2$ includes all the variables from the original problem with one additional variable λ. Here is the formulation

of $CP2$:

$$\min \quad \sum_{j\in E} c_j y_j + \sum_{l\in L} c_l y_l - c_L \lambda \tag{6.10}$$

$$(CP2) \quad \sum_{j\in E} y_j A_j + \sum_{l\in L} y_l A_l - \lambda e = 0 \tag{6.11}$$

$$\sum_{j\in E} \alpha_j y_j + \sum_{l\in L} \beta_l y_l = 1 \tag{6.12}$$

$$y \geq 0, \lambda \geq 0 \tag{6.13}$$

where $E = N \backslash L$. $c_L = \sum_{l\in L} c_l$ is the cost of the current solution. Positive y_j are entering variable and positive y_l are "staying" variable (not leaving the current solution). α_j and β_l are weights for the new normalization constraint.

Define P' as the set partitioning problem, having one extra dimension, defined on $x' = \begin{pmatrix} x \\ x_{n+1} \end{pmatrix}$, $c' = \begin{pmatrix} c \\ c_L \end{pmatrix}$, $A' = \begin{bmatrix} A & e \end{bmatrix}$, and $N' = N \cup \{n+1\}$; see formulation (6.1)-(6.3).

In P', we introduced an extra column (e) corresponding to the extra variable x_{n+1}. The extra column, by itself alone, is an aggregated artificial solution to P' (x_{n+1} set to 1 and all other x_j ($j \in N$) variables set to 0). Moreover, any solution to P is a reduced solution to P' ($x_{n+1} = 0$) and, hereinafter, refers to its non reduced form in P' as well.

The following remark is trivial when we see that the model (6.10)-(6.13) of $CP2$ above can be derived by applying the decomposition of ISUD to P', as in the formulation (6.4)-(6.7).

Remark 6.3.1. *$CP2$ is a complementary problem (call it CP' to mean: derived as CP from P') for ISUD on P' considering the aggregated column as the current integer solution (having $\begin{pmatrix} \alpha \\ \beta \\ 0 \end{pmatrix}$ as its normalization vector).*

So, any integer direction found by CP' will exchange the one single column from its current solution. In other words, the entering set of variables found by CP' replaces a solution to P and is then a solution to P. So, unlike CP, for which non null entering variables replace a part of the current solution that need to be changed to improve it, the non null variables in $CP2$ (non null y_j combined with non null y_l) form a complete and improved solution to P.

The following propositions characterize the solutions of the new model: their adjacency, optimality, relation to LP solutions and form.

Proposition 6.3.2. *All solutions to P', and so their equivalent solutions to P, are adjacent to the aggregated solution.*

Proof. In the complementary problem of ISUD, all non decomposable directions from the current solution lead to adjacent solutions (Theorem 3 from Balas & Padberg (1975)). In $CP2$, as CP', all directions are non decomposable (because any sub-direction is infeasible with regards to the compatibility constraints, so incompatible). So, they lead to adjacent solutions. Therefore, the solutions to P are adjacent to the current solution to P' which is the aggregated solution. □

Proposition 6.3.3. *There exists a vector* $\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$ *for which* $CP2$ *finds the optimal integer solution to* P.

Proof. As mentioned above, $CP2$ is seen as CP' for the particular one column aggregated current solution having c_L as cost. Lemma 9 from Rosat et al. (2016) proves the existence of a weight vector such that $CP2$ leads to the optimal adjacent solution, that is optimal by Proposition 6.3.2. So, knowing the right weight vector, $CP2$ is able to directly find an optimal solution of P in one single iteration. □

Proposition 6.3.4. *There is a biunivocal relation between the solutions of* LP *and the solutions of* $CP2$.

Proof. For every solution x to LP, define $\lambda = \sum_{j\in E}\alpha_j x_j + \sum_{l\in L}\beta_l x_l$ and $y = \lambda x$. $(y,\ \lambda)$ is a solution to $CP2$. For every solution $(y,\ \lambda)$ to $CP2$, $\lambda > 0$ (otherwise x is null). $\frac{1}{\lambda}y$ is a solution to LP. □

Proposition 6.3.5. *Define* $\|x\|_{\alpha,\beta} = \sum_{j\in E}\alpha_j x_j + \sum_{l\in L}\beta_l x_l$ *as the weighted* l_1*-norm of a solution* x *to* LP. *We have* $\|x\|_{\alpha,\beta} = \frac{1}{\lambda}$ *where* $(y,\ \lambda)$ *is the solution to* $CP2$ *corresponding to* x.

Proof. $y = \lambda \cdot x$ and the constraint (6.12) becomes $\lambda\|x\|_{\alpha,\beta} = 1$. This result can be derived also from Proposition 6.2.1 by considering the fact that $\lambda = \max_{l\in L}\lambda_l$. □

Corollary 6.3.6. *The objective function of* $CP2$ *becomes* $\lambda c\cdot x - \lambda c\cdot x^0 = \lambda c\cdot d = \frac{1}{\|x\|_{\alpha,\beta}} c\cdot d$.

So, the equivalent formulation of $CP2$ is:

$$(CP2) \quad \Leftrightarrow \quad \min \quad \frac{c\cdot(x-x^0)}{\|x\|_{\alpha,\beta}} \tag{6.14}$$

$$Ax = e \tag{6.15}$$

$$x \geq 0 \tag{6.16}$$

Remark 6.3.7. *$y = \frac{1}{\|x\|_{\alpha,\beta}} x$ is a normalized form of x.*

Proposition 6.3.8. *Directions found by $CP2$ might be decomposable with regard to the current solution x^0 to P.*

Proof. See the example in Sub-section 6.3.2. □

Remark 6.3.9. *Integer directions found by $CP2$ could be formed by a set of multiple non decomposable directions that CP would find separately in multiple iterations. This remark follows from Proposition 6.3.8 above.*

Proposition 6.3.10. *If (y, λ) is an integer solution (i.e., integer direction) to $CP2$, then $y_j > 0 \implies y_j = \lambda$, $j \in N$.*

Proof. $CP2$ being CP', this derives from Proposition 6 from Zaghrouti et al. (2014). □

6.3.2 New normalization weights

The main objective behind the normalization of the solution within ISUD is to encourage the integrality of the targeted solution. A well chosen normalization vector has the potential of minimizing the overlapping between the columns of the next solution. Under good conditions, like when the current solution presents good primal information (see Section 6 from Zaghrouti et al. (2014)), the normalization would lead to a solution with no overlap between its columns; thus integer.

The different normalization-weight coefficients studied by Rosat et al. (2016), including the one from the standard algorithm (Zaghrouti et al. (2014)), are the following (where $j \in I$, $l \in L$ and ω the normalization-weight vector for CP):

- $\omega_j = 1$; $\omega_l = 0$: favors directions with fewer entering columns.
- $\omega_j = 1$; $\omega_l = 1$: favors directions with fewer overall number of columns (entering + exiting).
- $\omega_j =$ the l_1-norm of column j; $\omega_l = 0$: favors directions with entering columns having smaller total number of non-zero components.
- $\omega_j =$ the degree of incompatibility of column j; $\omega_l = 0$: favors directions presenting fewer incompatibilities between their entering and exiting columns.

As mentioned above, the goal of using these ways of normalization is to minimize overlapping within the solution. In fact, solutions with less columns tend to have less overlapping; while many columns with large non-zero components tend to get overlapped within any solution. Also, the incompatibility between an entering column and an exiting one means that the exiting column needs more entering columns to cover the same rows. It may also mean that the

entering column needs to span over more exiting columns. So, higher incompatibility degrees within a solution would tend to increase the size of the solution and, then, its overlapping.

For I^{2}SUD, the same reasoning could be done about minimizing overlapping; but we need to take into consideration that I^{2}SUD finds directions that are decomposable and tend to be large in size (see Table 6.3 in Section 6.4). So, to increase the chances of having large integer directions, we would like to target large composite (decomposable) directions while encouraging sub-directions to remain small. For this, we set for $CP2$:

— $\alpha_j = 1 + \gamma_j$, $j \in E$; where γ_j is the number of columns, from the current integer solution, that are "broken" (their covered rows not fully covered) by the column corresponding to the variable x_j; $\beta_l = 1$, $l \in L$.

This setting of the norm (weighted l_1-norm of the direction) in $CP2$ tries to find a compromise between two ways of minimizing overlapping within the next solution:

— Minimizing the sum of γ_j over the entering columns diminishes the presence, within a sub-direction, of large highly incompatible columns that span over multiple exiting columns. We observed in the numerical results with ISUD (Rosat et al. (2016), Zaghrouti et al. (2014)) that a non decomposable direction (sub-direction of $CP2$) including columns with large γ_j contains generally a large number of non-zero variables and is not row-disjoint.

— The number of ones in α_j and β_l formulas sum up to the size of the next solution. An integer direction generally has a small number of nonzero variables. The entering variables replace a similar number of exiting variables. At the opposite, a fractional direction contains often more nonzero variables than the number of replaced variables. So, minimizing the number of ones in the norm has the effect of penalizing non integer sub-directions that enter more variables in the solution than the ones removed. Furthermore, it penalizes also the overlapping between rows covered by sub-directions. A direction with such overlapping enters more variables to replace the exiting ones.

We could consider that the non-decomposability aspect introduced with I^{2}SUD has added another dimension to the overlapping within directions found by $CP2$. In fact, the normalization weight vector of $CP2$ needs to address the local overlapping within sub-directions and the global overlapping between sub-directions within a composite direction. We can say that $CP2$ minimizes the ratio between the cost of the direction and an "over-estimation" of the size of the next solution.

The following example illustrates the use of the normalization weights, by both CP and $CP2$, for the choice of their best solution.

Example. Table 6.1 shows a problem P with 5 rows r_1, ..., r_5, 8 variables x_1, x_2, y_1, ...,y_6

and a known currentt integer solution $x_1 = 1$, $x_2 = 1$, $y_j = 0$ for all j. The cost vector is c. For CP, ω_j = the incompatibility degree for the column j ($j \in I$); and, for $CP2$, $\alpha_j = 1 + \gamma_j$ ($j \in E$) and $\beta_l = 1$ ($l \in L$). The table shows the solutions (sol) of LP with their respective costs (z). For both CP and $CP2$, it shows the direction (d), the norm of the direction ($\|d\|_w$), the normalized direction ($d^n = \dfrac{d}{\|d\|_w}$) and the cost of the normalized direction (z).

For LP, the table shows the three integer solutions other than the current one: (x_2, y_1, y_2), (y_1, y_2, y_5, y_6) and (x_1, y_5, y_6). The table also shows the optimal solution $(y_1, y_2, y_3, y_4, y_5)$ that is fractional.

CP presents three improving normalized non decomposable directions: (y_1, y_2), (y_3, y_4, y_5) and (y_5, y_6). (y_3, y_4, y_5) has the best reduced cost but is fractional. Note that we present only nonzero variables in the table.

For $CP2$, in addition to (y_1, y_2), (y_3, y_4, y_5) and (y_5, y_6), there are two other composite directions: $((y_1, y_2), (y_3, y_4, y_5))$ and $((y_1, y_2), (y_5, y_6))$. The best option is $((y_1, y_2), (y_3, y_4, y_5))$ and is, also, fractional; but it contains (y_1, y_2) which is integer and the latter can be extracted by the new algorithm (see Sub-section 6.3.3).

6.3.3 Improved algorithm

The general form of the algorithm for I^{2}SUD is as follows:

Step 0: Start with an initial integer solution.

Step 1: Considering the current integer solution, update $CP2$ (i.e., update the objective function and the normalization constraint).

Step 2:

- **a)** Solve $CP2$ to find an improving (i.e., descent) direction.
- **b)** If this direction is fractional, check if there is an extractable improving integer sub-direction.
- **c)** If an improving integer direction found in **a)** or sub-direction found in **b)**, update the current solution.

Control step:

- **a)** If Step 2 improves the solution, go to Step 1.
- **b)** If the gap is small enough, stop.
- **c)** Branch; if integer improving direction or sub-direction found, update the current solution and go to Step 1. Otherwise, stop.

As in ISUD, I^{2}SUD algorithm starts with an initial integer solution having, preferably, good primal information. Then, the algorithm iterates over Step 1, Step 2 and the Control step

until a stopping criterion is reached.

In Step 1, we set the cost of the current solution c_L in the objective function; and we update α_j and β_l, in the normalization constraint; they need to be dynamically changed from one iteration to another.

In Step 2, we solve $CP2$ for an improving direction. If the direction is fractional, the columns from, both, the current solution of P and the solution of $CP2$ (duplicate columns removed) are put in a MIP (a small SPP) to check for an improving integer sub-direction. If the direction or its sub-direction is integer, we update in c) the current solution with a simple exchange of columns.

Table 6.1 Example

			x_1	x_2	y_1	y_2	y_3	y_4	y_5	y_6	λ	$\|d\|_w$	z
		r_1	1		1								
		r_2	1			1							
		r_3		1			1		1				
		r_4		1				1	1				
		r_5		1			1	1		1			
		c	40	40	19	16	24	24	19	15			
LP	x_2,y_1,y_2	sol		1	1	1							
		z terms		40	19	16							75
	y_1,y_2,y_5,y_6	sol			1	1			1	1			
		z terms			19	16			19	15			69
	x_1,y_5,y_6	sol	1						1	1			
		z terms	40						19	15			74
	y_1,y_2,y_3,y_4,y_5	sol			1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$				
		z terms			19	16	$\frac{24}{2}$	$\frac{24}{2}$	$\frac{19}{2}$				68.5
		ω_j			1	1	2	1	1	1			
CP	(y_1,y_2)	d_j	-1		1	1							
		$w_j d_j$			1	1						2	
		d_j^n	-$\frac{1}{2}$		$\frac{1}{2}$	$\frac{1}{2}$							
		z terms	-$\frac{40}{2}$		$\frac{19}{2}$	$\frac{16}{2}$							-2.5
	(y_3,y_4,y_5)	d_j		-1			$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$				
		$w_j d_j$					1	$\frac{1}{2}$	$\frac{1}{2}$			2	
		d_j^n		-$\frac{1}{2}$			$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$				
		z terms		-$\frac{40}{2}$			$\frac{24}{4}$	$\frac{24}{4}$	$\frac{19}{4}$				-3.25
	(y_5,y_6)	d_j		-1					1	1			
		$w_j d_j$							1	1		2	
		d_j^n		-$\frac{1}{2}$					$\frac{1}{2}$	$\frac{1}{2}$			
		z terms		-$\frac{40}{2}$					$\frac{19}{2}$	$\frac{15}{2}$			-3
		$w : \beta_l$, $(1+\gamma_j)$	1	1	2	2	2	2	2	2			
$CP2$	(y_1,y_2)	d_j		1	1	1							
		$w_j d_j$		1	2	2						5	
		d_j^n		$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$					$\frac{1}{5}$		
		z terms		$\frac{40}{5}$	$\frac{19}{5}$	$\frac{16}{5}$					-$\frac{80}{5}$		-1
	(y_3,y_4,y_5)	d_j	1				$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$				
		$w_j d_j$	1				1	1	1			4	
		d_j^n	$\frac{1}{4}$				$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$		$\frac{1}{4}$		
		z terms	$\frac{40}{4}$				$\frac{24}{8}$	$\frac{24}{8}$	$\frac{19}{8}$		-$\frac{80}{4}$		-1.62
	(y_5,y_6)	d_j	1						1	1			
		$w_j d_j$	1						2	2		5	
		d_j^n	$\frac{1}{5}$						$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$		
		z terms	$\frac{40}{5}$						$\frac{19}{5}$	$\frac{15}{5}$	-$\frac{80}{5}$		-1.2
	(y_1,y_2),(y_3,y_4,y_5)	d_j			1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$				
		$w_j d_j$			2	2	1	1	1			7	
		d_j^n			$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$		$\frac{1}{7}$		
		z terms			$\frac{19}{7}$	$\frac{16}{7}$	$\frac{24}{14}$	$\frac{24}{14}$	$\frac{19}{14}$		-$\frac{80}{7}$		-1.64
	(y_1,y_2),(y_5,y_6)	d_j			1	1			1	1			
		$w_j d_j$			2	2			2	2		8	
		d_j^n			$\frac{1}{8}$	$\frac{1}{8}$			$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$		
		z terms			$\frac{19}{8}$	$\frac{16}{8}$			$\frac{19}{8}$	$\frac{15}{8}$	-$\frac{80}{8}$		-1.37

Within the Control step, if an improved integer solution is found in Step 2, then we go back to reiterate from Step 1. If no improving integer direction could be easily found, we check if the gap is satisfyingly small (see next section for gap computation). If this is the case, we stop. Notice that the ability to measure more accurately the gap, is one of the advantages that I^{2}SUD has over ISUD as discussed by Remark 6.3.14. Otherwise, we branch for an improving integer direction (see Remark 6.3.13 for a proposed branching method). If the branching is successful, we update the current solution and start over from Step 1. Otherwise, the algorithm terminates.

6.3.4 Remarks on I^{2}SUD

In this section, we present the main intrinsic differences between ISUD and I^{2}SUD. These differences do not include comparison of their performance and the quality of solutions they produce (see Section 6.4 for that).

Remark 6.3.11. *I^{2}SUD can make longer steps and, then, less iterations than ISUD.*

Given an initial integer solution, suppose the minimal path (sequence of adjacent integer solutions) to go from that initial solution to the optimal one is of length $k > 1$. Then, CP would need at least k iterations to reach optimality; since it can only reach the next adjacent solution in each iteration. $CP2$, on the other side, can reach optimality in less than k iterations (theoretically in one iteration as proved by Proposition 6.3.3) using composite directions.

When $CP2$ finds composite directions, it finds multiple sub-directions within the same iteration. To reach the same result, ISUD would make one iteration for each sub-direction. This behavior is confirmed by our experimentation.

Remark 6.3.12. *I^{2}SUD has better chances in finding improving integer directions; and needs less branching than ISUD.*

Composite fractional directions found by $CP2$ could still contain an improving integer sub-direction. So, when $CP2$ encounters a fractional direction, a small MIP is solved in Step 2 b) of I^{2}SUD to extract the best embedded improving integer sub-direction if there exists one. This is not possible in ISUD because the directions found by CP are non decomposable.

In other words, in contrast to a failure of CP, the failure of $CP2$ in finding an improving integer direction is not final and could be turned, by the small MIP, into a success if it finds an improving integer sub-direction (as illustrated in the example in Table 6.1). The

success of the small MIP in finding improving integer sub-directions was also confirmed by the numerical results.

Remark 6.3.13. *With I²SUD, we can implement a better branching schema.*

In Zaghrouti et al. (2014), the proposed branching method (see Section 3.3 from Zaghrouti et al. (2014)) is able to force a variable to be excluded from the solution of CP; but it is unable to force it to be part of the solution; and the variable remains free from branching constraints. The implemented version of this method explores only the "0-branch" where the whole fractional direction is excluded from the solution. In addition to this difficulty of not being able to force a column to be in the solution, a complete implementation of this method would not be very efficient because it would create many overlapping branches. And the worst drawback of this branching method is that, in the end, it does not guarantee to force the integrality of a solution of CP.

For $CP2$, with the introduction of the extra variable, we have the following: if columns $\{A_j | y_j > 0\}$ are disjoint, then y_j are all equal to each other and equal to λ (Proposition 6.3.10); otherwise, for certain j, we have $y_j < \lambda$. So, with I²SUD, it is easy to branch on a fractional variable y_j: to exclude it from the solution of $CP2$, we set $y_j = 0$ and to force it to take part of the solution, we set $y_j = \lambda$. In fact, the variable λ plays the role of the 1 in the formulation of P (in the right hand side and the integrality constraints).

So, this new branching method guarantees finding integer solutions if they exist and reaching optimality if fully implemented. Also, the branching can be stopped as soon as an integer sub-direction is found.

Remark 6.3.14. *With I²SUD, we can make a better estimation of the integrality gap.*

Both ISUD and I²SUD maintain upper bounds for the optimal solution. To know the gap for their best solution, they both need to have a lower bound. The easiest way of doing so is to solve the relaxation of the original problem and get the value of the relaxed solution. But, the targeted problems to solve are very large and solving their relaxation needs extra considerable time. If we exclude the option of solving the relaxation of the original problem, do the two algorithms have any other options for estimating the gap?

For ISUD, because of its multi-phase partial pricing, not all columns are present in CP and the solution of CP does not permit to estimate the gap for ISUD. Removing the partial pricing in ISUD is not an option because it is one of its main success factors.

For I²SUD, $CP2$ is successful without any partial pricing and contains all the columns from the original problem. So, we can obtain estimation for the gap after a given iteration.

Consider x^* the optimal solution to P and its corresponding value z^*. Denote by $\bar{z}$ the value of the current integer solution x^0 and by z_{CP2} the optimal value for $CP2$. So, we have by Corollary 6.3.6 and equivalence (6.14):

$z^* - \bar{z} = c \cdot x^* - c \cdot x^0 = \|x^*\|_{\alpha,\beta} \frac{1}{\|x^*\|_{\alpha,\beta}} (c \cdot x^* - c \cdot x^0) \geq \|x^*\|_{\alpha,\beta} z_{CP2}$.

So, setting $\|x^*\|_{\alpha,\beta}$ to its maximum possible value, say k, would give a loose lower bound. We will have: $|z^* - \bar{z}| \leq k |z_{CP2}|$. Setting it to some reasonable value (considering some knowledge about how would the optimal solution be; e.g., its maximum size (number of vehicles, drivers, pilots)) could give a good estimation for the lower bound.

Remark 6.3.15. *With I²SUD, we could consider problems with extra non set-partitioning constraints.*

We think that the greatest advantage of I²SUD is its ability to take into consideration supplementary non set-partitioning constraints. In fact, any non SPP constraint, in P, of the form $a_i \cdot x \leq b_i$ could be translated, in $CP2$, into $a_i \cdot y \leq \lambda b_i$ (where a_i and b_i are the coefficients and the right hand side element of the row i corresponding to the non SPP constraint). This way, I²SUD can, even, be generalized to any arbitrary binary problem. For this, some research and experimentation need to be done later.

6.4 Numerical results

For our experimentation, we compare I^2SUD to ISUD on the same instances solved in Zaghrouti et al. (2014). There are three sets of instances: Small (800x8900), Medium (1200x133000) and Large (1600x570000). Each set contains three groups: easy (50), medium (35) and difficult (20). Those numbers (50, 35 and 20) represent the percentage of columns from the optimal solution that remain present in the initial solution after a perturbation process (see Sub-section 6.1 from Zaghrouti et al. (2014)) and represent an increasing level of difficulty. Each group contains 10 instances for a total of 90. We tested I²SUD under the same conditions as ISUD.

Recall that the medium and large sets of instances are difficult to solve for CPLEX. Rosat et al. (2016) mentioned that, within a time limit of 10 hours, CPLEX could not find a feasible solution for the original problems of both sets.

We use a simplified version of the general algorithm described in Sub-section 6.3.3: the normalization weights are the new ones described in Sub-section 6.3.2. We use a heuristic stopping criterion instead of b) and c) of the Control step. The algorithm stops as soon as both $CP2$ and the MIP fail to improve the current solution. Actually, no branching is needed

because the test problems are solved to optimality without branching; the item **c)** from the Control step is completely disabled from the algorithm. The stopping criterion based on the gap estimated with the lower bound was also not needed because the optimal solutions are known; we are able to compare I^{2}SUD solutions to them.

We compare the results of our algorithm to the ones from Rosat et al. (2016) and Zaghrouti et al. (2014). Rosat et al. (2016) includes a comparison of four variants of ISUD and shows the best figures, in terms of quality of solution and performance (i.e., solution time), for each group of instances. The comparison criterion to select the best performer, ISUD(best), among all the variants tested in Rosat et al. (2016) was: Quality then performance (select the variant presenting the smallest gap; then, in case of a tie, select the fastest). The time to solve the three other variants is not included in the time of ISUD(best).

To check the quality of solution and performance (solution time) produced by our algorithm, we compare our results to the results of ISUD[2014] and the ones corresponding to the best cases for ISUD[2016] (see Table 6.2).

To show how different the two algorithms, I^{2}SUD and ISUD, are in terms of the number of iterations and the nature of directions found, we compare our results to those of ISUD[2014] (see Table 6.3). We do not compare with Rosat et al. (2016) becasue they did not include detailed information about the size of directions found.

Table 6.2 summarizes the results of the tests we ran on all instances and presents the average information for each group. From left to right, we show:

- Instance Group: the group of 10 instances: (S)mall, (M)edium or (L)arge followed by the level of difficulty 50, 35 or 20.
- Init. Err%: the initial gap between the initial integer solution and the optimal solution.
- For ISUD[2014], ISUD[2016] and I^{2}SUD
  - Time: the average time it takes the algorithm to solve an instance from the group.
  - Opt: the average time when the algorithm found its best integer solution.
  - Err%: the final gap between the best integer solution found and the optimal one.

Note that Rosat et al. (2016) ran their tests under an 8-core, 3.4 GHz, machine. We ran our tests on an i7, 2.7 GHz, laptop. Rosat et al. (2016) did not include results for the S50 group.

Table 6.2 ISUD vs. I^{2}SUD (quality and performance)

Instance	Init.	ISUD[2014] [1]			ISUD[2016]			I^{2}SUD		
Group	Err%	Time	Opt	Err %	Time	Opt	Err %	Time	Opt	Err %
S50	358.8	5.9	3.3	0.0	-	-	-	1.1	0.6	0.0
S35	466.9	6.7	4.3	0.1	7.1	4.4	0.0	0.9	0.6	0.0
S20	565.8	8.6	6.0	33.9	9.4	7.0	0.0	1.0	0.6	0.0
M50	52.0	74.6	38.8	0.0	70.0	27.0	0.0	16.2	6.7	0.0
M35	66.6	123.3	74.3	2.3	83.0	43.0	1.2	17.4	7.3	0.0
M20	82.2	132.1	79.6	16.0	93.0	62.0	5.5	17.9	7.6	0.0
L50	50.8	640.9	224.9	2.4	767.0	168.0	1.1	87.1	34.5	0.0
L35	65.6	803.8	350.5	5.2	924.0	277.0	0.0	88.2	35.0	0.0
L20	81.0	1443.4	709.1	5.6	1045.0	414.0	0.0	96.0	36.8	0.0

From Table 6.2, we notice that:

— I^{2}SUD is faster than ISUD: both times from I^{2}SUD (the total solution time and the time when finding the best solution) are smaller than the ones from ISUD. It is clear that the average overall performance gain is at least five times; and about ten times for large instances.

— I^{2}SUD reaches the optimal solution for all instances: I^{2}SUD outperforms the best variant of ISUD. Recall the selection criterion was based on the quality of solution first. So I^{2}SUD outperforms every variant of ISUD separately on quality of solution.

Table 6.3 compares the directions found by ISUD to those found by I^{2}SUD. It shows the same information for both algorithms. We find, from left to right:

— Instance Group: the group name.

— For both ISUD and I^{2}SUD

  — Total: the total number of directions found.

  — Disj: the number of integer directions among all the directions found.

  — Max: the maximum size of integer directions.

  — Avg: the average size of integer directions.

From Table 6.3, we notice that:

— I^{2}SUD makes a small number of iterations: For all 90 instances we solved, the maximum number of iterations was 3. The maximum number of improving iterations (iterations in which the algorithm finds an improving direction) was 2. For 82 instances out of 90, I^{2}SUD directly found the optimal solution in one single improving

1. We adjusted the times from Zaghrouti et al. (2014) (we divided by 1.67) to reflect how the performance would be under our test environment.

iteration. This was the case for 60 out of 60 instances from the medium and large groups.

— I^2SUD finds directions of larger size.

Table 6.3 ISUD vs. I^2SUD (iterations and directions)

Instance	ISUD				I^2SUD			
Group	Total	Disj	Max	Avg	Total	Disj	Max	Avg
S50	37.5	30.2	6.5	2.5	2.2	1.2	52.0	47.0
S35	41.5	33.3	9.5	2.9	2.4	1.4	67.4	54.6
S20	48.7	38.8	12	3.38	2.2	1.2	81.8	73.8
M50	16.4	14.7	3.6	2.19	2.0	1.0	25.3	25.3
M35	21.2	18.2	4.5	2.35	2.0	1.0	32.4	32.4
M20	20.8	18.6	4.8	2.58	2.0	1.0	38.5	38.5
L50	17.2	16.9	3.6	2.25	2.0	1.0	33.3	33.3
L35	23.7	23.1	4.9	2.39	2.0	1.0	43.0	43.0
L20	26.5	25.4	8.0	3.06	2.0	1.0	53.1	53.1

As a general observation, the two algorithms ISUD and I^2SUD present two opposite behaviors. The first finds small directions and makes more iterations while the second finds large directions and needs less iterations. This is due to the fact that ISUD is limited to finding non decomposable directions (which tend to be small) while I^2SUD has the freedom of finding composite directions (which are large). Both algorithms find paths from an initial solution to an optimal one; but, in terms of iterations, ISUD is obliged to take every atomic step in the path. I^2SUD, on the other side, is allowed to take shortcuts within its path.

6.5 Conclusion

In this paper, we improve the Integral Simplex Using Decomposition method. We introduce new model, algorithm and normalization weights. The new version, while being faster than the older one, reaches the best solutions for all the instances in our tests. In addition to being performing, its most important advantage is that it opens the possibilities of its extension to arbitrary binary problems instead of remaining specific to set partitioning problems. Also, we prove that our method is able to find the optimal solution for any SPP by solving one single LP if we have the right normalization weights. We were able to find normalization weights permitting to solve some hard vehicle and crew scheduling instances in less than three iterations. This may encourage and push the research towards ideas leading to finding "perfect" CP models for other classes of set partitioning problems.

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans le cadre de cette thèse, nous présentons trois travaux de recherche qui se complètent pour le développement d'un algorithme qui résout efficacement les problèmes de partitionnement de grande taille. Plus particulièrement, nous avons effectué nos tests sur des instances de problèmes de rotations d'équipages (chauffeurs d'autobus et pilotes d'avions). L'algorithme, mis au point par ces travaux, arrive à cerner trois facteurs qui impactent la performance de la résolution du problème de partitionnement selon un paradigme primal.

Sur un premier plan, l'algorithme est efficace contre la dégénérescence. La dégénérescence est un phénomène intrinsèque au problème de partitionnement. Les premiers algorithmes à vouloir proposer des méthodes primales pour la résolution du problème de partitionnement ont particulièrement souffert de ce phénomène. Notre algorithme traite efficacement ce phénomène en utilisant, itérativement, une décomposition du problème en deux sous-problèmes plus faciles à résoudre : le problème réduit est non dégénéré et le problème complémentaire trouve des directions de descente.

Sur un deuxième plan, moyennant une décomposition dynamique, la charge liée au contrôle de l'intégralité est transférée au problème réduit. Le problème complémentaire trouve une direction de descente et le problème réduit se charge de trouver une solution dans un voisinage autour de cette direction de descente. Cette dernière indique une zone d'amélioration potentielle où il est plus probable de trouver facilement une solution entière de meilleur coût. Ceci se fait d'une façon efficace en déléguant le branchement au niveau du problème réduit au solveur commercial.

Sur un troisième plan, l'algorithme réduit le nombre d'itérations nécessaires (ou le nombre de solutions intermédiaires à visiter) avant d'atteindre la solution optimale. Ceci se fait en relâchant, dans le nouveau modèle du troisième article, l'aspect "adjacence" exigé par le modèle initial du premier article. Ainsi, la nouvelle version de notre algorithme est capable de chercher des solutions subséquentes non adjacentes qui présentent de bonnes améliorations au niveau du coût. Cela a pour effet d'aller, plus loin, vers des solutions de coût beacoup plus intéressant que les solutions voisines adjacentes. L'intégralité est favorisée avec une contrainte de normalisation adaptée.

Les tests de l'algorithme effectués sur des instances de grande taille de problèmes d'horaires de chauffeurs d'autobus confirment que l'algorithme est efficace. Pour ces instances, une solution optimale est trouvée dans l'espace de quelques minutes alors que ce sont des instances considérées très difficiles pour CPLEX qui demande beaucoup plus de temps pour les

résoudre.

Les idées introduites dans les articles 2 et 3 pourraient être intégrées dans un même algorithme. L'article 3 introduit une méthode efficace pour trouver avec le problème complémentaire des directions entières d'amélioration de la solution entière. Il y réussit dans la plupart des cas. Dans les cas où le problème complémentaire ne trouve pas de direction entière, l'article 2 propose une méthode efficace pour trouver une solution entière améliorée dans le problème réduit. En combinant les idées de ces articles, on pourrait obtenir un algorithme encore robuste et plus performant.

Bien que nous avons effectué nos tests sur des problèmes de partitionnement (purs), les problèmes avec contraintes supplémentaires fréquemment rencontrés en industrie peuvent, en principe, être résolus par l'approche proposée. Nous pensons, donc, avoir proposé un algorithme qui supporte une large variété de problèmes de type partitionnement. Les tests et les adaptations nécessaires ainsi que l'intégration dans un contexte de génération de colonnes sont à planifier dans le cadre de projets de recherche futures.

Enfin, nous signalons que, dès les premiers résultats de notre premier travail, plusieurs pistes de recherche ont été ouvertes. Cela a fait que différents projets de recherche ont vu le jour. En particulier, nous citons : l'intégration de notre algorithme avec la génération de colonnes et sa parallélisation.

CHAPITRE 8 CONCLUSION ET RECOMMANDATION

Dans cette thèse, nous avons proposé un algorithme efficace pour la résolution du problème de partitionnement. L'algorithme proposé est une méthode constructive qui y trouve une séquence de solutions entières à coûts décroissants convergeant vers une solution optimale entière.

Notre algorithme est capable de résoudre des problèmes d'horaires d'équipages (chauffeurs d'autobus, pilotes d'avions) de grande taille selon un paradigme primal et dans des temps très compétitifs. Il arrive à résoudre en quelques minutes des problèmes nécessitant plusieurs heures avec les méthodes de résolution actuelles. Nous trouvons qu'avec ce résultat nous avons atteint l'objectif tracé pour cette thèse.

Le problème de partitionnement étant très utilisé en industrie, nous pensons que le travail effectué dans le cadre de cette thèse sera aussi d'une utilité importante. Nous croyons que l'intégration de notre travail dans un contexte de génération de colonnes aura des retombées économiques intéressantes.

Nous pensons que les développements théoriques et algorithmiques proposés devront changer la pratique quant à la résolution des problèmes de partitionnement. En particulier, le fait de pouvoir pénaliser les directions menant aux solutions fractionnaires est une innovation majeure. Notre approche se généralise bien au cas de problème de partitionnement avec contraintes supplémentaires et ouvre de nouvelles pistes pour des projets de recherche futures.

RÉFÉRENCES

E. Balas & M. W. Padberg, “On the set-covering problem”, *Operations Research*, vol. 20, no. 6, pp. 1152–1161, 1972.

——, “On the set-covering problem : II. an algorithm for set partitioning”, *Operations Research*, vol. 23, no. 1, pp. 74–90, 1975.

——, “Set partitioning : A survey”, *SIAM review*, vol. 18, no. 4, pp. 710–760, 1976.

M. L. Balinski & R. E. Quandt, “On an integer program for a delivery problem”, *Operations Research*, vol. 12, no. 2, pp. 300–304, 1964.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, & P. H. Vance, “Branch-and-price : Column generation for solving huge integer programs”, *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

H. Bouarab, I. Elhallaoui, A. Metrane, & F. Soumis, “Dynamic constraint and variable aggregation in column generation”, *Cahiers de GERAD*, no. G-2014-82, 2014.

J.-F. Cordeau, G. Desaulniers, N. Lingaya, F. Soumis, & J. Desrosiers, “Simultaneous locomotive and car assignment at VIA Rail Canada”, *Transportation Research Part B : Methodological*, vol. 35, no. 8, pp. 767–787, 2001.

F. H. Cullen, J. J. Jarvis, & H. D. Ratliff, “Set partitioning based heuristics for interactive routing”, *Networks*, vol. 11, no. 2, pp. 125–143, 1981.

M. Desrochers & F. Soumis, “A column generation approach to the urban transit crew scheduling problem”, *Transportation Science*, vol. 23, no. 1, pp. 1–13, 1989.

M. Desrochers, J. Desrosiers, & M. Solomon, “A new optimization algorithm for the vehicle routing problem with time windows”, *Operations Research*, vol. 40, no. 2, pp. 342–354, 1992.

I. Elhallaoui, D. Villeneuve, F. Soumis, & G. Desaulniers, “Dynamic aggregation of set-partitioning constraints in column generation”, *Operations Research*, vol. 53, no. 4, pp. 632–645, 2005.

I. Elhallaoui, A. Metrane, F. Soumis, & G. Desaulniers, “Multi-phase dynamic constraint aggregation for set partitioning type problems”, *Mathematical Programming*, vol. 123, no. 2, pp. 345–370, 2010.

I. Elhallaoui, A. Metrane, G. Desaulniers, & F. Soumis, “An improved primal simplex algorithm for degenerate linear programs”, *INFORMS Journal on Computing*, vol. 23, no. 4, pp. 569–577, 2011.

M. Gamache, F. Soumis, G. Marquis, & J. Desrosiers, “A column generation approach for large-scale aircrew rostering problems”, *Operations research*, vol. 47, no. 2, pp. 247–263, 1999.

J. B. Gauthier, J. Desrosiers, & M. E. Lübbecke, “Tools for primal degenerate linear programs : IPS, DCA, and PE”, *Cahiers de GERAD*, no. G-2013-51, 2013.

——, “About the minimum mean cycle-canceling algorithm”, *Discrete Applied Mathematics*, vol. 196, pp. 115–134, 2015.

——, “Vector space decomposition for linear programs”, *Cahiers de GERAD*, no. G-2015-26, 2015.

R. E. Gomory, “Outline of an algorithm for integer solutions to linear programs”, *Bulletin of the American Society*, vol. 64, pp. 275–278, 1958.

——, “An all-integer integer programming algorithm”, *Industrial Scheduling*, J. F. Muth & G. L. Thompson, éds. Prentice Hall, Englewood Cliffs, N.J., 1963, pp. 193–206.

K. Haase, G. Desaulniers, & J. Desrosiers, “Simultaneous vehicle and crew scheduling in urban mass transit systems”, *Transportation Science*, vol. 35, no. 3, pp. 286–303, 2001.

U.-U. Haus, M. Köppe, & R. Weismantel, “The integral basis method for integer programming”, *Mathematical Methods of Operations Research*, vol. 53, no. 3, pp. 353–361, 2001.

K. L. Hoffman & M. Padberg, “Solving airline crew scheduling problems by branch-and-cut”, *Management Science*, vol. 39, no. 6, pp. 657–682, 1993.

A. N. Letchford & A. Lodi, “Primal cutting plane algorithms revisited”, *Mathematical Methods of Operations Research*, vol. 56, no. 1, pp. 67–81, 2002.

M. E. Lübbecke & J. Desrosiers, “Selected topics in column generation”, *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2005.

A. Metrane, F. Soumis, & I. Elhallaoui, “Column generation decomposition with the degenerate constraints in the subproblem”, *European Journal of Operational Research*, vol. 207, no. 1, pp. 37–44, 2010.

V. Raymond, F. Soumis, A. Metrane, & J. Desrosiers, “Positive edge : A pricing criterion for the identification of non-degenerate simplex pivots”, *Cahiers de GERAD*, no. G-2010-61, 2010.

V. Raymond, F. Soumis, & D. Orban, “A new version of the improved primal simplex for degenerate linear programs”, *Computers & Operations Research*, vol. 37, no. 1, pp. 91–98, 2010.

E. Rönnberg & T. Larsson, “Column generation in the integral simplex method”, *European Journal of Operational Research*, vol. 192, no. 1, pp. 333–342, 2009.

S. Rosat, I. Elhallaoui, F. Soumis, & A. Lodi, “Integral simplex using decomposition with primal cuts”, *Experimental Algorithms*. Springer, 2014, pp. 22–33.

S. Rosat, I. Elhallaoui, F. Soumis, & D. Chakour, “Influence of the normalization constraint on the integral simplex using decomposition”, *Discrete Applied Mathematics*, 2016. DOI : `http://dx.doi.org/10.1016/j.dam.2015.12.015`

A. Saxena, “Set-partitioning via integral simplex method”, Thèse de doctorat, Ph. D. thesis, Carnegie Mellon University, Pittsburgh, 2003.

G. L. Thompson, “An integral simplex algorithm for solving combinatorial optimization problems”, *Computational Optimization and Applications*, vol. 22, no. 3, pp. 351–367, 2002.

V. Trubin, “On a method of solution of integer linear programming problems of a special kind”, *Soviet Mathematics Doklady*, vol. 10, 1969, pp. 1544–1546.

R. D. Young, “A simplified primal (all-integer) integer programming algorithm”, *Operations Research*, vol. 16, no. 4, pp. 750–782, 1968.

A. Zaghrouti, I. El Hallaoui, & F. Soumis, “Improving ILP solutions by zooming around an improving direction”, *Cahiers de GERAD*, no. G-2013-107, 2013.

A. Zaghrouti, F. Soumis, & I. El Hallaoui, “Integral simplex using decomposition for the set partitioning problem”, *Operations Research*, vol. 62, no. 2, pp. 435–449, 2014.