UNIVERSITÉ DE MONTRÉAL

MÉTHODES PRIMALES POUR RÉSOUDRE LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES DE RESSOURCES

ILYAS HIMMICH
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
OCTOBRE 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MÉTHODES PRIMALES POUR RÉSOUDRE LE PROBLÈME DE PLUS COURT CHEMIN AVEC CONTRAINTES DE RESSOURCES

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

M. GENDREAU Michel, 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. BEN AMOR Hatem, Ph. D., membre et codirecteur de recherche
M. DESAULNIERS Guy, Ph. D., membre
M. MEDAGLIA Andrés L., Ph. D., membre externe

CITATION

« Celui qui abandonne son foyer pour se mettre en quête du savoir suit la voie de Dieu ... L'encre du savant est plus sacrée que le sang du martyr. »

LE PROPHÈTE MUHAMMAD

DÉDICACE

À ma mère Fatima, ma fierté, mon amour inné, maman qui a dédié sa vie, et qui a sacrifié ses désirs rien que pour le bien-être de ses enfants.

...

À mon père Ali, l'arbre solide et droit qui m'inspire, c'est pour toi papa que je dessine mes passions, car sans toi, je n'aurais pas été moi, je t'aime.

...

À ma reine Hind, c'est grâce à toi que ma vie a un sens, j'ai de la chance d'être avec toi, je t'aime mon amour.

...

À mon futur ange Sami que j'attends avec impatience, j'ai hâte de te voir mon chéri, je t'aime.

...

À mes deux frères Marouane et Safouane, je suis fier de vous, vous faites partie de moi.

...

REMERCIEMENTS

Après avoir rendu grâce à Allah le tout-puissant, le tout Miséricordieux, Il m'est agréable de m'acquitter d'une dette de reconnaissance auprès de toutes les personnes, dont l'intervention au cours de cette thèse, a favorisé son aboutissement.

Ainsi, j'exprime ma profonde gratitude à Issmaïl El Hallaoui, qui m'a ouvert les portes du GERAD pour pouvoir accomplir cette thèse. Je le remercie tout d'abord pour l'honneur qu'il m'a fait en me confiant la réalisation de ce projet. Je le remercie également pour ces encouragements tout au long de cette thèse. Il était prêt à tout moment à conseiller, soulager et semer de l'espoir même pendant les périodes les plus ardues que nous avons vécues ensemble. Sa disponibilité, ses qualités humaines, son sens humaniste, ses conseils éclairés et son judicieux encadrement se sont tous réunis pour me permettre d'accomplir le présent travail.

Mes remerciements les plus sincères vont également à François Soumis, mon codirecteur de recherche, d'abord pour la confiance qu'il m'a accordée en m'offrant cette opportunité d'être codirigé par un expert de son rang. Je le remercie ensuite pour le soutien financier qu'il m'a garanti pendant tout mon parcours au doctorat. Enfin, je le remercie pour ses remarques pertinentes, sa grande rigueur et ses précieux conseils qui étaient incontestablement d'une utilité importante et dont j'ai assez bénéficié.

Je remercie également Hatem Ben Amor pour avoir accepté de codiriger ce travail. Je le remercie pour sa collaboration pendant les deux premières années et pour les leçons précieuses de vie qu'il m'a apprises.

Je tiens à exprimer ma gratitude à Mohamed Ouzineb, mon professeur à l'Institut National de Statistique et d'Économie Appliquée au Maroc. Je lui suis particulièrement reconnaissant pour m'avoir bâti le premier pont qui m'a permis de faire un doctorat au Canada.

J'adresse des sincères remerciements à Abdelouahab Zaghrouti pour son assistance informatique et pour les conseils techniques qu'il m'a prodigués. Des remerciements spéciaux vont à Adil Tahir, mon premier colocataire avec qui j'ai passé de merveilleux moments. Je remercie chaleureusement Omar Foutlane qui a partagé avec moi le même parcours, et qui m'a fait bénéficier généreusement de son expérience en vie.

Aussi, je n'oublierai jamais les autres collègues de travail avec qui j'ai réussi à créer un environnement de fraternité et de collaboration. Ainsi, je remercie Rachid Hassani, Amine Amrouss, Safae Er-Rbib, Abderrahman Bani, Mayssoun Messaoudi, Issoufou Abdou Amadou et Salah-eddine Makhloufi.

J'exprime ma gratitude à tous les professeurs qui ont assuré ma formation au doctorat : François Soumis, Guy Desaulniers, Alain Hertz, Charles Audet et Pierre Hansen.

Que messieurs les membres de jury trouvent ici l'expression de ma reconnaissance pour avoir accepté de juger mon travail.

Que tous ceux et celles qui ont contribué de près ou de loin à l'accomplissement de ce travail trouvent l'expression de mes remerciements les plus chaleureux.

RÉSUMÉ

Le problème de plus court chemin avec contraintes de ressources consiste à trouver un chemin entre deux nœuds dans un réseau (une source et une destination) à un coût minimum tout en respectant des contraintes sur la consommation de ressources. Il s'agit d'une généralisation du problème classique du plus court chemin non contraint.

Ce problème a été largement étudié dans la littérature. Nous l'utilisons particulièrement comme sous-problème lors de la résolution des problèmes de planification de tournées de véhicules et d'horaires d'équipages par un algorithme de génération de colonnes. L'approche standard pour résoudre le problème de plus court chemin avec les contraintes de ressources est la programmation dynamique. Cette méthode est une extension du fameux algorithme de Bellman-Ford qui prend en considération les contraintes de ressources. Elle consiste à construire une séquence de sous-chemins provenant du nœud source en étendant ceux existants aux nœuds successeurs à l'aide d'une fonction de prolongation. Chaque sous-chemin correspond à un état et est reconnu par une étiquette qui mémorise son coût et ses consommations de ressources. La fonction de prolongation assure l'élimination des étiquettes non réalisables et garantit la mise à jour des coûts et des consommations de ressources après chaque prolongation. Des règles de dominance sont également utilisées pour interdire l'extension d'étiquettes peu prometteuses.

D'un côté, cette approche est capable de gérer des règles complexes de travail provenant des conventions collectives et des mesures de sécurité et qui sont généralement non linéaires et même non convexes. D'un autre côté, la méthode de programmation dynamique permet de générer de nombreuses solutions réalisables (chemins réalisables) au lieu d'une seule, ce qui est nécessaire dans un contexte de génération de colonnes. Cependant, lorsqu'il faut gérer un grand nombre de ressources, le nombre d'étiquettes augmente de manière exponentielle, notamment dans le cas de réseaux de grande taille avec des centaines de milliers d'arcs. Par conséquent, le processus de résolution nécessite beaucoup de temps et dans de nombreux cas, nous ne sommes pas en mesure de trouver des solutions optimales. Plusieurs heuristiques ont été proposées pour gérer cette situation ; certaines dominent sur un sous-ensemble de ressources sélectionnées de manière empirique, alors que d'autres se contentent de prolonger un sous-ensemble d'étiquettes de chaque nœud. Bien évidemment, n'étant pas fondées mathématiquement, ces méthodes n'offrent aucune garantie sur la qualité des solutions retournées.

Nous proposons dans ce travail différentes idées qui sont capables de remédier aux inconvénients mentionnés ci-dessus, afin d'améliorer la résolution du problème de plus court chemin

avec les contraintes de ressources. Les méthodes proposées sont primales, exactes et tirent profit des avantages de la programmation dynamique.

La première contribution de cette thèse est un nouvel algorithme primal multi-directionnel appelé *MultiDirectional Dynamic Programming Algorithm*. L'approche proposée partitionne l'espace d'états en petits sous-espaces disjoints qui sont explorés séquentiellement dans plusieurs itérations. Nous proposons aussi de nouvelles techniques d'apprentissage qui permettent à cet algorithme de tirer profit des résultats des itérations précédentes, afin de réduire la dimension des sous-espaces subséquents et générer rapidement de meilleurs chemins. Les expérimentations numériques sur des instances du problème de planification de tournées de véhicules et d'horaires d'équipages avec plus de 600.000 nœuds et 1.000.000 arcs démontrent que la nouvelle approche vainc l'algorithme standard de programmation dynamique. En particulier, elle est capable de générer des chemins réalisables avec jusqu'à 90% du coût optimal en moins de 10% du temps requis par l'algorithme standard de programmation dynamique.

Étant convaincus de l'efficacité de l'exploration itérative de l'espace d'état, nous proposons dans une seconde contribution un autre algorithme primal exact appelé *Primal Adjacency-Based algorithm*. Nous fournissons d'abord une nouvelle étude polyédrique qui nous permet d'introduire une nouvelle partition de l'espace des états basée sur la notion d'adjacence. L'algorithme proposé utilise cette partition pour explorer de manière itérative l'espace d'états et produit une séquence d'ensembles de chemins réalisables de coûts non décroissants. Ces chemins sont ensuite utilisés pour enrichir l'information primale disponible, ce qui permet d'accélérer le processus de résolution dans les itérations suivantes. Les expérimentations numériques sur les mêmes instances citées ci-dessus montrent d'excellentes performances de cet algorithme. Il est capable, à l'instar de l'algorithme multi-directionnel, de produire des chemins de très bonne qualité dans des délais très courts. De plus, il réduit considérablement le nombre d'étiquettes créées par rapport à l'algorithme standard de programmation dynamique et à l'algorithme multi-directionnel.

Les résultats obtenus ont montré que les approches proposées constituent des outils de résolution très efficaces, parfaitement adaptées à la méthode de génération de colonnes. Pour cette raison, nous nous concentrons dans notre troisième contribution sur le développement d'un nouveau cadre de résolution appelé *Primal Column Generation Framework* qui intègre ces méthodes primales dans un schéma de génération de colonnes. Ceci permet de trouver rapidement et intelligemment les colonnes de coûts réduits négatifs nécessaires en résolvant une séquence de sous-problèmes restreints en fonction des besoins. De plus, ce paradigme primal confère à la génération de colonnes une autonomie et une grande flexibilité. Des résultats expérimentaux montrent que l'outil proposé est capable de trouver des solutions optimales

tout en réduisant le temps consommé à résoudre les sous-problèmes par des facteurs allant jusqu'à 7 fois par rapport à un algorithme de génération de colonnes standard. Cela engendre des gains significatifs en matière du temps total de résolution avec un facteur de réduction moyen de 3.5.

ABSTRACT

The shortest path problem with resource constraints is to find a path between two nodes in a network (a source and a sink) at minimum cost while respecting constraints on resource consumption. This problem is a generalization of the classical non constrained shortest path problem.

This problem has been largely studied in the literature. We particularly use it as a subproblem to solve crew scheduling and vehicle routing problems by the column generation method. The standard approach to solve the shortest path problem with resource constraints is dynamic programming. This method is an extension of the well-known Bellman-Ford algorithm that takes into account the resource constraints. It constructs a sequence of subpaths originated from the source node, by extending the existing ones to the successor nodes. Each subpath corresponds to a state and is recognized using a label that stores its cost and its resource consumptions. The extension function ensures the elimination of infeasible labels and guarantees the update of costs and resource consumption after each extension. Dominance rules are also used to prohibit the extension of unpromising labels.

This approach is able to handle complex working rules like collective agreement rules and other safety rules that may be nonlinear and even non convex. Also, it allows the generation of many feasible solutions (feasible paths) instead of one, which is required in a column generation context. However, when we have to deal with a large number of resources, the number of labels increases exponentially, especially in the case of huge networks of hundreds of thousands of arcs. Consequently, the solution process becomes time consuming and in many cases we are not able to find optimal solutions. Several heuristics have been proposed to handle this situation, some of them dominate on an empirically selected subset of resources, while others used to extend only limited subsets of labels from each node. Of course, given that these methods are not mathematically founded, they offer no guarantee on the quality of the returned solutions.

We propose in this work different ideas that are able to handle the drawbacks mentioned above, in order to improve the resolution of the shortest path problem with resource constraints. The proposed methods are primal, exact and take profits from the advantages of dynamic programming.

The first contribution of this thesis is a new primal algorithm called the *MultiDirectional Dynamic Programming Algorithm*. The proposed approach splits the state space into small disjoint subspaces that are sequentially explored in several iterations. Moreover, we propose

new learning techniques that allow the proposed algorithm to build on the results of the previous iterations, to reduce the dimension of the subsequent subspaces and to quickly generate better paths. Numerical experiments on Vehicle and Crew Scheduling Problem instances with up to 600.000 nodes and 1.000.000 arcs demonstrate that the new approach outperforms the standard dynamic programming algorithm. In particular, the multidirectional algorithm is able to generate feasible paths with up to 90% of the optimal cost in less than 10% of the time required by standard dynamic programming.

Being convinced of the efficiency of the iterative exploration of the state space, we propose in a second contribution another exact primal algorithm called *Primal Adjacency-Based algorithm*. We first provide a new polyhedral study that allows us to introduce a new path adjacency-based partition of the state space. The proposed algorithm uses this partition to iteratively explore the state space and produces a sequence of sets of feasible paths of non decreasing costs. These paths are used in order to enrich the available primal information which improve the solution process in the subsequent iterations. Computational experiments on the same instances cited above show the excellent performance of this algorithm. Similarly to the multidirectional algorithm, the Primal Adjacency-Based algorithm is able to produce very interesting paths in very limited portions of time. Moreover, it drastically reduces the number of created labels compared to both standard dynamic programming and multidirectional algorithms.

The obtained results have shown that the proposed approaches provide a highly efficient solution tool, nicely suitable for the column generation method. For this reason, we focus in our third contribution on developing a new *Primal Column Generation* framework that embeds these primal methods inside a column generation scheme. This framework allows finding quickly and intelligently the required negative reduced costs columns by solving a sequence of restricted subproblems as needed. Furthermore, this primal paradigm endows the column generation with a self-acting ability and a large degree of flexibility. Computational experiments show that the proposed tool is able to find optimal solutions while reducing the time spent solving subproblems by factors up to 7 times. This yields significant gains in the total solution times with an average reduction factor of 3.5 compared to the standard column generation algorithm.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

LISTE DES SIGLES ET ABRÉVIATIONS

CSPP	Constrained Shortest Path Problem
RCSPP	Resource Constrained Shortest Path Problem
SPPTW	Shortest Path Problem with Time Windows
SPPRC	Shortest Path Problem with Resource Constraints
SP	SubProblem
RSP	Restricted SubProblem
MP	Master Problem
RMP	Restricted Master Problem
VCSP	Vehicle and Crew Scheduling Problem
CG	Column Generation
DP	Dynamic Programming
MDDPA	MultiDirectional Dynamic Programming Algorithm
NF	Nearest First
BF	Best First
LSP	Label Storing Procedure
LLS	Label Loading Strategy
LLEL	Learning from Locally Efficient Labels
FDD	Feasible Descent Direction
IDD	Improving Descent Direction
ACD	Average Cost Decrease
PAB	Primal Adjacency-Based
ILA	Improved Labeling Algorithm
DCL	Dominance Calls per Label
LRF	Label Reduction Factor
PCG	Primal Column Generation
CCP	Control Component Procedure

CHAPITRE 1 INTRODUCTION

Dans un contexte économique hyper concurrentiel, marqué essentiellement par la globalisation de l'activité économique, les entreprises manifestent de plus en plus un énorme besoin de planification et d'optimisation de toutes leurs activités, en l'occurrence, l'approvisionnement en matières premières, le processus de production et la distribution des produits finis. Par conséquent, la gestion des ressources de transport demeure une activité principale au cœur de la chaîne logistique globale, assurant le déplacement des personnes et l'acheminement des biens et services au bon endroit, en bonne quantité et au bon moment. Une raison pour laquelle la logistique de transport tient une place majeure dans les décisions des entreprises.

Afin de répondre à ce besoin croissant du milieu industriel, beaucoup de recherches mathématiques ont été dédiées aussi bien au développement de méthodes de planification des activités de transport, qu'au perfectionnement des réseaux routiers, maritimes et aériens. Le problème de tournées de véhicules et d'horaires d'équipages se trouve au cœur des intérêts de la logistique du transport. C'est un problème d'optimisation très fréquent autant dans le transport des personnes que dans la distribution et la collecte des produits. Étant donné une flotte de véhicules, un ensemble de chauffeurs (ou pilotes) et un ensemble de points à visiter (clients, stations ou aéroports), une version simplifiée du problème consiste à déterminer un ensemble de tournées permettant de couvrir, à moindre coût, tous les points à visiter. Ces tournées varient selon le domaine d'application. Elles peuvent être des tournées de collecte ou de livraison de biens, de transport de personnes, d'intervention (maintenance, réparation, contrôle), de visites (visites médicales, commerciales, etc.) ou même des rotations de vols.

Cette version simplifiée du problème de tournées de véhicules et d'horaires d'équipages est généralement utilisée pour des raisons théoriques et académiques. En effet, le problème devient plus riche en pratique avec l'ajout d'une grande variété de contraintes servant à modéliser les problématiques des compagnies de transport, notamment la limitation de la capacité des véhicules, les restrictions relatives aux horaires de travail des chauffeurs ainsi que les contraintes de fenêtres de temps qui exigent de visiter chaque point de collecte ou de livraison dans un intervalle de temps bien déterminé. Les contraintes peuvent aussi imposer des restrictions sur le nombre de quarts de travail, le nombre de vols ou bien le nombre de tâches à effectuer par véhicule. Ces restrictions ont donné naissance à de nouvelles versions plus ardues du problème de tournées de véhicules telles que le problème de tournées de véhicules avec contraintes de ressources et le problème de tournées de véhicules avec fenêtres de temps.

Au sein du Groupe d'Études et de Recherche en Analyse de Décisions (GERAD), les re-

cherches dans la logistique de transport ont vu le jour à la fin des années 1970. Le fruit incontesté de ces travaux était le logiciel GENCOL. Ce logiciel est dédié à la résolution des problèmes de tournées de véhicules et d'horaires d'équipages en utilisant une approche de résolution connue sous le nom de *génération de colonnes* (Column Generation - CG). La méthode de CG est une approche itérative dédiée principalement à la résolution des problèmes d'optimisation de grande taille. Cette méthode est basée sur la décomposition de Dantzig & Wolfe (1960) qui permet de scinder ces problèmes en un *problème maître* (Master Problem - MP) et un ou plusieurs *sous-problèmes* (SubProblems - SPs). Cette décomposition exige que le problème à résoudre ait une structure particulière. En particulier, il doit comporter deux classes de contraintes : des contraintes locales qui sont séparables par sous-ensembles de variables et des contraintes globales qui lient les différentes variables du problème. Ainsi, le MP considère uniquement les contraintes globales, alors que les contraintes locales sont déléguées aux SPs.

D'un côté, le MP est le plus souvent un problème de partitionnement ou de recouvrement d'ensemble, avec probablement des contraintes supplémentaires (Haase *et al.*, 2001). D'un autre côté, les SPs correspondent à des instances du problème de plus court chemin avec contraintes de ressources (Shortest Path Problem with Resource Constraints - SPPRC) ; ils servent à nourrir le MP par des chemins réalisables de coûts réduits négatifs. Ces derniers sont introduits dans le MP sous forme de colonnes en leur associant de nouvelles variables.

Un SPPRC est défini sur un réseau connexe orienté $G(V, A)$, il consiste à chercher le chemin de coût minimum entre un nœud source et un nœud destination dans G qui respecte certaines restrictions. Ces dernières sont modélisées à l'aide des contraintes de ressources. La définition des ressources dépend du contexte d'application, il s'agit d'une mesure de quantités telles que le temps de parcours, le temps de repos, la charge d'un véhicule, le nombre de quarts de travail, ou bien le nombre de clients à visiter. En fait, en plus du coût, chaque arc du réseau est muni d'un vecteur de consommations de ressources qui désigne les valeurs consommées de chaque ressource lors du parcours de cet arc. Ainsi, chaque sous-chemin du nœud source à un nœud $i \in V$ est caractérisé par un coût et un vecteur mémorisant la consommation cumulée de chaque ressource. Les contraintes de ressources sont imposées à chaque nœud du réseau pour chaque ressource sous forme d'intervalles appelés *fenêtres de ressources*. Dans le cas général, la valeur cumulée de chaque ressource le long d'un sous-chemin donné est restreinte à varier entre les bornes supérieure et inférieure de la fenêtre de ressource associée.

Certes, le SPPRC est une généralisation du problème classique de plus court chemin non contraint. Cependant, l'ajout des contraintes de ressources rend ce dernier NP-dur dans le sens large (Dumitrescu & Boland, 2003) même pour le cas d'un graphe acyclique avec des

coûts positifs. Ce résultat a été prouvé par Handler & Zang (1980) pour le cas d'une seule contrainte de ressource. De plus, le problème qui consiste à vérifier l'existence d'un chemin réalisable est NP-complet dès lors que le nombre de ressources considérées est supérieur ou égal à deux (Laval *et al.*, 2006).

De ce fait, il n'existe pas d'algorithmes polynomiaux capables de résoudre de façon optimale le SPPRC même pour le cas d'une seule contrainte de ressource. Dans une grande partie des problèmes de tournées de véhicules et d'horaires d'équipages, faisant appel à la méthode de CG, il faut résoudre à chaque itération des dizaines de SPs, dont chacun est une instance de SPPRC. Le nombre d'itérations peut atteindre un millier pour les problèmes de grande taille. Il est donc impératif dans ce contexte de résoudre des dizaines de milliers de SPPRC. C'est ainsi que les SPs consomment la plus grande portion du temps total de la résolution par CG. Par conséquent, l'efficacité de cette dernière est très dépendante de la pertinence des méthodes utilisées pour résoudre les SPs. Un investissement dans cette classe de problèmes s'impose bien évidemment au premier rang des intérêts des entreprises offrant des services de planification de tournées de véhicules et d'horaires d'équipages.

Le SPPRC est apparu pour la première fois dans la thèse de doctorat de Desrochers (1986) comme SP lors de la résolution par CG du problème de planification des horaires de chauffeurs de bus. Depuis lors, plusieurs recherches se sont penchées sur le développement de méthodes efficaces pour la résolution du SPPRC et de ses variantes.

Les solveurs commerciaux utilisant la méthode de CG font appel à la programmation dynamique (Dynamic Programming - DP) pour résoudre les SPs. Le premier algorithme de type programmation dynamique a été proposé par Desrochers & Soumis (1988a). Il s'agit d'une généralisation du fameux algorithme de Bellman. Cette approche associe au nœud source un sous-chemin vide et essaie d'étendre ce dernier dans toutes les directions possibles. Ceci donne naissance, en chaque nœud $i \in V$, à un ensemble de sous-chemins allant du nœud source à i. Chaque sous-chemin représente un état et il est identifié par un vecteur, nommé *étiquette*, qui mémorise le coût et les consommations cumulées de chacune des ressources le long des arcs composant ce sous-chemin. Les extensions des sous-chemins se font à l'aide d'une fonction de prolongation qui vérifie également la réalisabilité des sous-chemins vis-à-vis des contraintes de ressources. Ainsi, l'algorithme arrête l'extension des sous-chemins non réalisables et élimine immédiatement leurs étiquettes associées. Les algorithmes de DP utilisent également des règles de dominance basées sur la notion d'optimalité au sens de *Pareto*. Ces règles ont pour but d'établir un ordre de préférence partiel entre les étiquettes appelé *ordre de Pareto*. Cet ordre permet de détecter les étiquettes moins intéressantes afin de les élaguer sans influencer l'optimalité de l'algorithme. Bien évidemment, l'efficacité de ces algorithmes

dépend principalement de leur capacité de réduire la dimension de l'espace des états (espace des étiquettes) en éliminant le maximum possible d'étiquettes.

L'importance des algorithmes de DP revient principalement à leur aptitude à surmonter une multitude de complexités du SPPRC. D'un côté, ces approches traitent efficacement les contraintes de ressources qui peuvent être non linéaires et même non convexes. D'un autre côté, elles sont capables de produire plusieurs solutions entières, ce qui dissimule tout souci d'intégralité. Néanmoins, lorsque le nombre de ressources considérées est grand, les règles de dominance deviennent faiblement applicables. Par conséquent, un nombre exponentiel d'étiquettes est créé, ce qui donne naissance à un espace d'états de taille énorme. Devant cette situation, les méthodes exactes utilisant la DP deviennent relativement fastidieuses et très coûteuses en matière de temps de calcul.

Ces difficultés ont motivé les chercheurs à réfléchir à des méthodes heuristiques capables de retourner de bonnes solutions en un temps raisonnable. Parmi les méthodes proposées, on cite la dominance sur un sous-ensemble de ressources, l'introduction d'une borne supérieure sur le nombre d'étiquettes à conserver dans chaque nœud ou bien la projection des vecteurs de ressources sur un espace de dimension inférieure au nombre de ressources (Nagih & Soumis, 2006). Ces heuristiques ont pu améliorer le processus de résolution par DP en offrant différents niveaux de performance liés aussi bien à la qualité de la solution qu'au temps requis pour la résolution. Toutefois, toutes ces stratégies exigent d'être ajustées et validées régulièrement pour chaque problème et parfois pour chaque classe d'instances, en plus, elles n'offrent aucune garantie sur la qualité de la solution.

Notre travail s'inscrit dans le cadre général de la résolution par CG des problèmes d'horaires d'équipages (chauffeurs de bus, pilotes d'avions, ...) de grande taille. Dans ce contexte, la résolution des SPs vise à générer le plus rapidement possible des colonnes de coûts réduits négatifs (pour les problèmes de minimisation) afin d'améliorer la solution courante du MP. C'est dans cette optique que nous nous intéressons à développer de nouvelles méthodes de résolution du SPPRC afin d'accélérer le processus de résolution des SPs dans un contexte de CG. Les approches que nous nous proposons de développer doivent répondre à 3 exigences : 1. être capables de tirer profit des avantages de la DP, 2. être capables de remédier aux faiblesses de cette classe de méthodes, 3. être adaptées aux exigences de la méthode de CG, à savoir pouvoir générer des solutions réalisables de très bonne qualité en un temps très limité. Pour ce faire, nous optons pour des méthodes de type *primal*. Contrairement aux méthodes duales qui sont incapables de générer une solution primale réalisable avant la fin de la résolution, les méthodes primales effectuent une série de recherches locales dans le domaine réalisable. Cette caractéristique permet de produire une séquence de solutions primales réalisables au cours de

la résolution avant de converger à une optimale. Nous notons que les travaux proposés dans cette thèse présument l'acyclicité des réseaux puisque c'est le cas des réseaux espaces-temps fréquemment utilisés en pratique.

Dans une première partie de ce travail, nous proposons un nouvel algorithme multi-directionnel du type DP appelé *"MultiDirectional Dynamic Programming Algorithm"* (MDDPA). Il est connu que les algorithmes standards de DP sont des algorithmes unidirectionnels dans le sens que la recherche se fait dans une seule direction à partir d'une étiquette initiale au nœud source. Une amélioration de l'algorithme DP, appelée *"programmation dynamique bidirectionnelle"* a été proposée par Righini & Salani (2006). Cette méthode consiste à propager les étiquettes à la fois en avant, de la source à la destination, et en arrière, de la destination à la source. Le MDDPA est une généralisation des recherches unidirectionnelle et bidirectionnelle. L'idée de la recherche multi-directionnelle consiste à munir ces algorithmes d'une structure leur permettant d'effectuer des recherches séquentielles sur des espaces de recherche de taille réduite. À cette fin, nous introduisons une nouvelle formulation généralisée du SPPRC. Cette dernière suppose que chaque nœud du réseau est muni d'un ensemble d'étiquettes réalisables, au lieu de considérer une seule étiquette au nœud source comme c'est le cas pour la formulation classique. Cette structure permet de subdiviser l'espace des états en plusieurs sous-espaces disjoints dont chacun peut être exploré indépendamment des autres. L'extension de chaque sous-ensemble d'étiquettes définit une direction de recherche dans un sous-espace d'états. L'algorithme MDDPA consiste en premier lieu à préparer les ensembles d'étiquettes à attribuer aux nœuds du réseau. Ensuite, il étend ces ensembles de façon séquentielle conformément à une stratégie de recherche prédéfinie.

Nous proposons deux stratégies de recherche différentes : *Nearest first* et *Best first*. La première stratégie donne la priorité aux étiquettes les plus proches du nœud destination, vu que ces dernières requièrent un effort de calcul très limité avant de générer des chemins complets au nœud destination. Cette sélection d'étiquettes est basée sur la distance des étiquettes par rapport au nœud destination. Cette distance est mesurée par le biais d'une nouvelle notion de *Cocycle* qui, de sa part, s'appuie sur l'acyclicité des réseaux considérés. Comme deuxième stratégie, nous prolongeons en premier lieu les étiquettes ayant les coûts réduits les plus négatifs qui promettent la génération de chemins de très bonne qualité. Ces étiquettes sont choisies de partout dans le réseau peu importe leurs distances du nœud destination. Finalement, nous dotons l'algorithme proposé par des techniques d'apprentissage qui lui permettent de tirer profit de l'information générée au cours des itérations précédentes afin d'accélérer les résolutions subséquentes. Notamment, le coût du meilleur chemin courant est utilisé pour serrer les bornes du coût sur les nœuds du réseau afin de réduire la taille des sous-espaces d'états non encore explorés. Les étiquettes précédemment générées sont également utilisées pour fortifier

l'élimination des étiquettes non prometteuses nouvellement créées. En particulier, nous utilisons les étiquettes efficaces parmi celles disponibles afin de définir des directions de descente. Ces directions correspondent à des séquences d'arcs qui, une fois ajoutées à des sous-chemins, assurent la construction rapide de nouveaux chemins réalisables et améliorants. Un chemin est dit *améliorant* si son coût réduit est meilleur que celui du meilleur chemin disponible.

Afin d'évaluer expérimentalement la méthode proposée, nous avons opté pour des instances de SPs extraites de différentes itérations de la CG lors de la résolution du problème d'horaires de véhicules et d'équipages (Vehicle and Crew Scheduling Problem - VCSP) (voir Haase *et al.* (2001)). L'algorithme MDDPA a montré un très bon comportement numérique comparé à un algorithme standard de DP offert par la librairie BOOST de C++. En effet, il a pu réduire le temps de calcul par un facteur de 3 à 5. Le nombre d'étiquettes a été réduit de plus de 3 fois. En plus, l'algorithme a montré sa capacité de produire des chemins réalisables de coûts réduits avec 90 % de la valeur optimale en moins de 10 % du temps requis par la DP. Ce résultat répond largement aux besoins de la méthode de CG.

Convaincus de l'efficacité de l'algorithme MDDPA, nous avons décidé d'intensifier la recherche dans la même direction, à savoir la conception d'algorithmes du type primal. Comme deuxième contribution, nos recherches se sont focalisées sur le développement de nouveaux outils permettant d'effectuer des recherches rapides sur des espaces de dimension réduite menant à l'optimalité. C'est dans cette optique que nous avons mené une étude polyédrique du SPPRC. La recherche effectuée était fructueuse dans la mesure où elle a donné naissance à un nouvel algorithme plus pertinent que l'algorithme MDDPA appelé : *"Primal Adjacency-Based algorithm"* (PAB). Il s'agit d'une méthode itérative exacte basée essentiellement sur la notion d'*adjacence* au niveau du polyèdre du domaine réalisable.

Étant donné un point initial composé d'un ensemble de chemins, l'algorithme PAB explore le voisinage de ce point de façon itérative et génère par conséquent une séquence d'ensembles de chemins réalisables de coûts réduits négatifs convergeant à une solution optimale. Le voisinage de ce point est défini grâce à un degré d'adjacence qui permet d'établir un partitionnement de l'ensemble de chemins en sous-ensembles disjoints. Pour construire le point initial, nous faisons appel à l'information primale qui est souvent disponible a priori pour chaque problème sous forme de résultats des résolutions anciennes, comme elle peut être extraite de la structure du problème en question. L'algorithme utilise également une technique de stockage dynamique des étiquettes qui l'empêche de créer une étiquette plus d'une fois. Une autre contribution qu'apporte cette méthode est la technique de combinaison. Cet outil permet de fabriquer des chemins améliorants ayant des degrés d'adjacence élevés uniquement en combinant les chemins précédemment générés lors de la recherche dans les voisinages de

degrés inférieurs.

L'algorithme PAB a été évalué sur les mêmes instances du VCSP utilisées par l'algorithme MDDPA. D'un côté, il est largement plus efficace que la DP standard. D'un autre côté, il converge à une solution optimale plus rapidement que l'algorithme MDDPA pour toutes les instances, tout en restant très compétitif à ce dernier au niveau du temps requis pour prouver l'optimalité.

En résumé, les deux approches proposées ont montré un grand potentiel en vue de la résolution à l'optimalité des instances de SPPRC. Par ailleurs, étant des méthodes primales, ces approches produisent séquentiellement des ensembles de chemins réalisables de coûts réduits non croissants qui convergent à l'optimalité. Ces chemins sont en général de très bonne qualité et sont générés dans des délais très courts. Cet aspect s'adapte énormément bien aux exigences de la méthode de CG dont l'objectif ultime est de nourrir le MP aussi vite que possible par des colonnes de bonne qualité. Pourtant, il s'avère difficile de prévoir le comportement de ces méthodes dans un contexte global de CG où la qualité des valeurs duales retournées par le MP influence la complexité des SPs.

Afin de vérifier dans quelle mesure les effets secondaires dus aux valeurs duales peuvent nuire au bon fonctionnement de nos méthodes primales, nous proposons comme troisième contribution d'intégrer ces deux méthodes dans un contexte de CG. C'est ainsi que nous développons un nouveau cadre intitulé : *"Primal Column Generation"*, il s'agit d'un outil de résolution parfaitement adapté à la nature primale de MDDPA et PAB. Le PCG se base sur les décompositions de l'espace d'états introduites par MDDPA et PAB afin de définir des sous-problèmes restreints (Restricted SubProblems - RSPs) de taille réduite. Ces derniers sont résolus de façon itérative au besoin tout en permettant au solveur d'apprendre des résultats des itérations passées. D'un côté, le PCG fait profiter la CG de la puissance de ces deux méthodes pour résoudre les SPs. D'un autre côté, il dote le solveur du SP d'un grand degré de flexibilité qui lui permet d'arrêter la résolution des RSPs une fois que des solutions jugées satisfaisantes sont retournées. Les résultats expérimentaux sur des instances de VCSP ont confirmé l'efficacité locale de MDDPA et PAB pour résoudre les SPs au sein du PCG. De plus, cette efficacité locale a engendré une performance globale qui s'est clairement manifestée au niveau du temps total de résolution. Ce dernier a été réduit par un facteur moyen de 3.5.

La présente thèse est organisée comme suit : tout d'abord, nous situons le lecteur au *chapitre 2* dans le contexte général de notre travail. Nous y décrivons le problème général de tournées de véhicules et d'horaires d'équipages, avant de présenter les fondements théoriques de la méthode de CG. Une revue de littérature est présentée au *chapitre 3*, nous y rappelons

les différentes variantes du SPPRC, les approches proposées en vue de le résoudre avant de faire un survol sur les applications du problème. Dans le *chapitre 4*, nous présentons la problématique à laquelle le présent travail se propose de faire face, nous y traçons également les grandes lignes des améliorations proposées.

Le *chapitre 5* est consacré à notre première contribution intitulée : *"A Multidirectional Dynamic Programming Algorithm for the Shortest Path Problem with Resource Constraints"*. Ensuite, nous présentons au *chapitre 6* notre deuxième travail, à savoir : *"A Primal Adjacency-Based Algorithm for the Shortest Path Problem with Resource Constraints"*. Nous présentons au *chapitre 7* notre troisième papier intitulé : *"Primal Column Generation framework for solving the Vehicle and Crew Scheduling Problem"*. En conclusion, nous faisons au *chapitre 8* une synthèse globale de nos contributions, les différentes extensions de nos travaux de recherche ainsi que des propositions de nos perspectives de recherche.

CHAPITRE 2 CONTEXTE GÉNÉRAL

La présente thèse concerne principalement au SPPRC. Ce problème apparaît en pratique comme SP lors de la résolution des problèmes de tournées de véhicules et d'horaires d'équipages avec la méthode de CG. Il s'agit d'un cadre général permettant de modéliser une grande famille d'applications qui peuvent être rencontrées dans divers environnements tels que le transport, la fabrication, l'entreposage et les services, etc. (Desaulniers *et al.*, 1998b). Un point commun entre ces problèmes est qu'ils visent tous à couvrir un ensemble de tâches prédéterminées tout en minimisant les coûts d'opération.

Dans ce chapitre, nous présentons d'abord une formulation générique du problème général en question. Plus loin, nous expliquons formellement les bases théoriques de la décomposition de Dantzig & Wolfe (1960) sur laquelle se fonde la méthode de CG. Une description du mécanisme de fonctionnement de cette méthode clôt le chapitre.

2.1 Formulation générique

Étant donné un ensemble de véhicules et d'employés et un ensemble de tâches H, les problèmes de tournées de véhicules et d'horaires d'équipages sont souvent modélisés à l'aide de réseaux espace-temps. Ils consistent à définir un ensemble de chemins (ou horaires) permettant de couvrir, à moindre coût, l'ensemble de tâches H. D'un côté, chaque chemin de la solution doit vérifier des contraintes locales telles que les contraintes de connexité ou d'élémentarité de chemins et les contraintes de ressources imposées par les règles de gestion. D'un autre côté, ces chemins sont soumis à des restrictions globales, liées principalement à la disponibilité des véhicules, des employés, à la couverture des tâches ou aux limitations dues aux relations potentielles de couplage ou de préséance.

Le premier modèle général du problème de tournées de véhicules et d'horaires d'équipages a été introduit par Desrosiers *et al.* (1995). Ce modèle, bien qu'il modélise un grand éventail d'applications, est incapable de supporter des exigences assez complexes qui apparaissent en pratique. Une formulation plus générale appelée *formulation unifiée* a été proposée par Desaulniers *et al.* (1998b). Cette formulation prend en considération les différentes contraintes qui peuvent s'imposer dans les applications réelles.

Afin de décrire la formulation unifiée, nous considérons le cas du problème générique multi-commodités où chaque véhicule (ou employé) définit une commodité. L'ensemble de commodités étant K, à chaque commodité $k \in K$ est associé un réseau $G^k(V^k, A^k)$. Dans ce

réseau, $V^k = N^k \cup \{o(k), d(k)\}$ est l'ensemble de noeuds comprenant le noeud source $o(k)$ et le noeud destination $d(k)$ relatifs à la commodité k et A^k est l'ensemble d'arcs qui peuvent être parcourus par la commodité k. En plus du coût c^k_{ij}, on associe à chaque arc $(i,j) \in A^k$ un vecteur de consommation de ressources $r^k_{ij} = \{r^{k1}_{ij}, r^{k2}_{ij}, ..., r^{k|\mathcal{R}^k|}_{ij}\}$ où $\mathcal{R}^k$ est l'ensemble de ressources associées à la commodité $k \in K$.

Dans ce qui suit, toutes les variables du problème sont indexées par l'indice de la commodité $k \in K$ à laquelle elles sont associées. Ainsi, $X^k = \{X^k_{ij} | (i,j) \in A^k\}$ est l'ensemble de variables de flot binaires telles que $X^k_{ij} = 1$ si l'arc $(i,j) \in A^k$ est parcouru par la commodité k ; 0 sinon. Les cumuls des ressources sur les sous-chemins allant d'une source $o(k)$ et se terminant à un noeud $i \in V^k$ sont dénotés $R^k_i = \{R^{kt}_i | i \in V^k, t \in \mathcal{R}^k\}$, et engendrent des coûts c^{kt}_i. On utilise f^{kt}_{ij} pour désigner la fonction de prolongation de la ressource $t \in \mathcal{R}^k$ du noeud i au noeud j pour la commodité k. Cette fonction est décrite dans la section 2.2.2.

En plus des variables de flot X^k et de ressources R^{kt}_i, la formulation unifiée fait appel à des variables supplémentaires $Y_s, s \in S$ qui apparaissent dans les contraintes liantes afin de modéliser différentes situations industrielles. Ces variables sont utilisées pour compter le nombre de commodités utilisées par une solution. Elles peuvent servir également, selon les applications, comme variables d'écart ou de surplus auxquelles sont associées des pénalités c_s dans la fonction objectif. De plus, la formulation unifiée supporte l'existence d'un ensemble de contraintes liantes W. On désigne par p_{hs} et q_{ws} les coefficients des variables $Y_s, s \in S$ dans les contraintes de couverture de tâches $h \in H$ et dans les contraintes liantes relatives à W respectivement. De même, on associe à chaque variable de flot des coefficients $q^k_{w,ij}$ qui désignent la contribution de chaque variable dans l'ensemble de contraintes liantes W et des constantes binaires $p^k_{h,ij}$ qui valent 1 si l'arc $(i,j) \in A^k$ couvre la tâche $h \in H$; 0 sinon. Les contributions des variables de ressources dans les contraintes liantes sont dénotées $q^{kt}_{w,i}$. Finalement, on dénote par L^k un ensemble de contraintes particulières pour une commodité $k \in K$. Les coefficients $d^k_{l,ij}$, $d^{kt}_{l,i}$ et d_l sont respectivement les coefficients des variables de flot, celles des ressources et le membre de droite dans chaque contrainte particulière $l \in L^k$.

Le modèle unifié correspond au programme non linéaire mixte suivant :

$$Min \quad \sum_{k\in K}\sum_{(i,j)\in A^k} c_{ij}^k X_{ij}^k + \sum_{k\in K}\sum_{i\in V^k}\sum_{t\in\mathcal{R}^k} c_i^{kt} R_i^{kt} + \sum_{s\in S} c_s Y_s \tag{2.1}$$

sous les contraintes :

$$\sum_{k\in K}\sum_{(i,j)\in A^k} p_{h,ij}^k X_{ij}^k + \sum_{s\in S} p_{h,s} Y_s = p_h \quad \forall\, h \in H \tag{2.2}$$

$$\sum_{k\in K}\sum_{(i,j)\in A^k} q_{w,ij}^k X_{ij}^k + \sum_{k\in K}\sum_{i\in V^k}\sum_{t\in\mathcal{R}^k} q_{w,i}^{kt} R_i^{kt} + \sum_{s\in S} q_{w,s} Y_s = q_w \quad \forall\, w \in W \tag{2.3}$$

$$l_s \leq Y_s \leq u_s, \forall\, s \in S \tag{2.4}$$

$$\sum_{j:(i,j)\in A^k} X_{ij}^k - \sum_{j:(j,i)\in A^k} X_{ji}^k = \begin{cases} -1 & \text{for } i = o(k) \\ 0 & \forall\ i \in N^k, \quad \forall\, k \in K \\ 1 & \text{for } i = d(k) \end{cases} \tag{2.5}$$

$$X_{ij}^k (f_{ij}^{kt}(R_i^k, r_{ij}^{kt}) - R_j^{kt}) \leq 0 \quad \forall\, k \in K, \forall\, t \in \mathcal{R}^k, \forall\, (i,j) \in A^k \tag{2.6}$$

$$a_i^{kt} \,(\sum_{j\in V^k} X_{ij}^k) \leq R_i^{kt} \leq b_i^{kt} \,(\sum_{j\in V^k} X_{ij}^k) \quad \forall\, k \in K, \forall\, t \in \mathcal{R}^k, \forall\, i \in V^k \tag{2.7}$$

$$\sum_{(i,j)\in A^k} d_{l,ij}^k X_{ij}^k + \sum_{i\in V^k}\sum_{t\in\mathcal{R}^k} d_{l,i}^{kt} R_i^{kt} \leq d_l \quad \forall\, k \in K, \forall\, l \in L^k \tag{2.8}$$

$$X_{ij}^k \in \{0,1\} \quad \forall\, k \in K, \forall\, (i,j) \in A^k \tag{2.9}$$

La fonction objectif consiste à minimiser la somme de trois termes : 1. les coûts de parcours des arcs par l'ensemble des commodités, 2. les coûts générés par les différentes ressources suite à l'extension des sous-chemins composant les chemins de la solution, 3. les pénalités et les coûts fixes engendrés par les variables supplémentaires.

Les contraintes (2.2) sont les contraintes de couverture de tâches. Elles imposent que chaque tâche $h \in H$ soit couverte p_h fois. Les contraintes de type (2.3) sont des contraintes liantes plus générales que les contraintes de couverture des tâches. L'utilité de ces contraintes se définit selon l'application en question. Des exemples sur l'utilité de ce genre de contraintes

ont été énumérés par Desaulniers *et al.* (1998b). Les contraintes (2.4) définissent les valeurs admissibles des variables supplémentaires $Y_s, s \in S$.

Les contraintes (2.5) à (2.9) sont séparables par commodité. Les contraintes (2.5) sont des contraintes de conservation de flot, elles garantissent la connexité d'un chemin entre $o(k)$ et $d(k)$ associé à chaque commodité $k \in K$. Les contraintes (2.6) et (2.7) sont des contraintes de ressources. Plus de détails sur ces contraintes sont donnés dans la section 2.2.2. Les contraintes (2.8) servent à imposer des exigences particulières propres à chaque commodité. Il s'agit d'une forme générique qui est capable de modéliser diverses restrictions telles que l'élimination des sous-tours pour le cas de réseaux cycliques, les relations de couplage ou de préséance. Finalement, les contraintes (2.9) sont des contraintes d'intégralité des variables de flot.

2.2 Décomposition de Dantzig-Wolfe

Les problèmes de tournées de véhicules et d'horaires d'équipages, comme pour la plupart des problèmes classés NP-complets, présentent des difficultés lors de la résolution des instances de grande taille. Les méthodes classiques de programmation linéaire en nombres entiers permettent de résoudre de façon exacte certains de ces problèmes ayant une taille relativement réduite. Cependant, ces méthodes perdent leur efficacité face aux problèmes de grande taille. L'approche de CG demeure la meilleure méthode pour ce genre de problèmes. Cette méthode est basée sur une décomposition appelée *décomposition de Dantzig-Wolfe* (Dantzig & Wolfe, 1960).

La formulation unifiée distingue entre deux classes de contraintes : la première classe est formée de contraintes globales qui lient les différentes variables du problème associées à toutes les commodités, à savoir les contraintes de couverture (2.2), les contraintes liantes (2.3) ainsi que les contraintes de bornes (2.4) sur les variables supplémentaires $Y_s, s \in S$; la deuxième classe est composée de contraintes locales qui se séparent par commodité. Il s'agit principalement de contraintes de connexité de chemins et de contraintes de ressources associées à chaque commodité. Cette classe de contraintes forment une structure de blocs angulaire décrite dans la figure 2.1. Le principe de Dantzig & Wolfe (1960) permet de décomposer la formulation unifiée, en un MP et $|K|$ SPs, un SP par commodité.

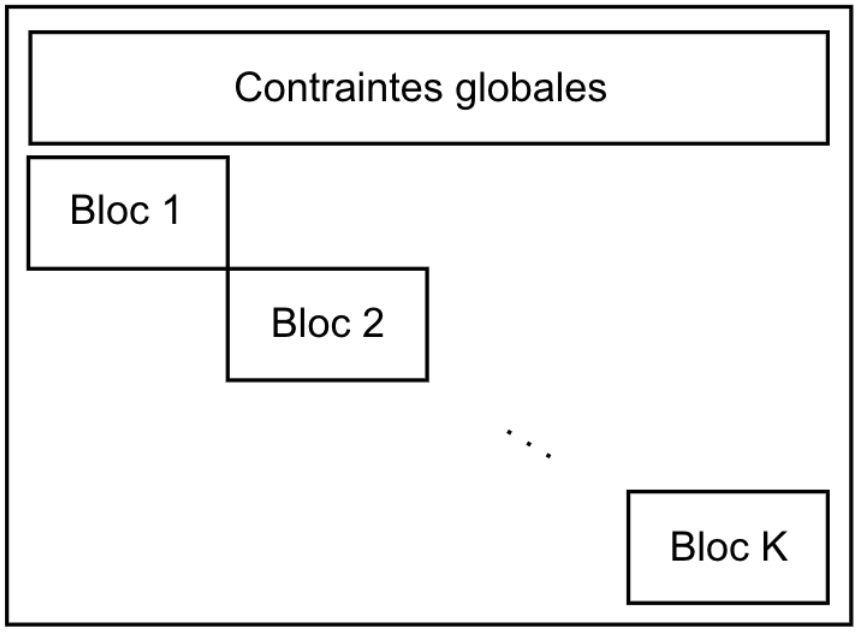

Figure 2.1 Structure de blocs angulaire

2.2.1 Problème maître

Le MP est une reformulation du problème original qui considère uniquement les contraintes globales du problème. Les variables du MP correspondent à des points extrêmes des SPs.

Soit δ^k l'espace de solutions du SP associé à la commodité k. Les contraintes de conservation de flot (2.5) permettent d'envoyer une unité de flot de $o(k)$ à $d(k)$. Ces contraintes définissent alors une structure de chemin soumis en outre à des contraintes de ressources. Les points extrêmes des SPs sont alors des chemins réalisables dans G^k. On note Ω^k l'ensemble de points extrêmes du SP associé à la commodité k et on associe à chaque chemin $\pi \in \Omega^k$ une variable θ_π^k appelée *variable de chemin*.

La formulation du MP découle du théorème de *Minkowski-Weyl* qui s'énonce comme suit :

Théorème 1. *Soit δ un espace de solutions et conv(δ) son enveloppe convexe. Un point $x \in$ conv(δ) si et seulement s'il peut s'écrire comme combinaison convexe des points extrêmes de conv(δ) plus une combinaison linéaire non négative de ses rayons extrêmes.*

Étant bornés, les espaces $\delta^k, k \in K$ ne possèdent pas de rayon extrême. Par conséquent, toute solution (X_{ij}^k, R_i^k) du SP k s'écrit simplement comme combinaison convexe des points extrêmes décris par les vecteurs de flot et de ressource suivants :

$$(x^k_\pi, \tau^k_\pi) = (x^k_{ij,\pi}, \tau^{kt}_{ij,\pi}), \;\; k \in K, \pi \in \Omega^k, (i,j) \in A^k, t \in \mathcal{R}^k \tag{2.10}$$

Lesdites combinaisons s'écrivent :

$$X^k_{ij} = \sum_{\pi \in \Omega^k} x^k_{ij,\pi} \theta^k_\pi \qquad \forall\, k \in K, \forall\, (i,j) \in A^k \tag{2.11}$$

$$X^k_{ij} \in \{0,1\} \qquad \forall\, k \in K, \forall\, (i,j) \in A^k \tag{2.12}$$

$$R^{kt}_i = \sum_{\pi \in \Omega^k} \tau^{kt}_{i,\pi} \theta^k_\pi \qquad \forall\, k \in K, \forall\, t \in \mathcal{R}^k, \forall\, i \in V^k \tag{2.13}$$

$$\sum_{\pi \in \Omega^k} \theta^k_\pi = 1 \qquad \forall\, k \in K \tag{2.14}$$

$$\theta^k_\pi \geq 0 \qquad \forall\, k \in K, \forall\, \pi \in \Omega^k \tag{2.15}$$

Finalement, ce changement de variables permet de réécrire le MP comme suit :

$$Min \quad \textstyle\sum_{k \in K} \sum_{\pi \in \Omega^k} c^k_\pi \theta^k_\pi + \sum_{s \in S} c_s Y_s \tag{2.16}$$

sous les contraintes :

$$\sum_{k \in K} \sum_{\pi \in \Omega^k} p^k_{h,\pi} \theta^k_\pi + \sum_{s \in S} p_{h,s} Y_s = p_h \qquad \forall\, h \in H \tag{2.17}$$

$$\sum_{k \in K} \sum_{\pi \in \Omega^k} q^k_{w,\pi} \theta^k_\pi + \sum_{s \in S} q_{w,s} Y_s = q_w \qquad \forall\, w \in W \tag{2.18}$$

$$l_s \leq Y_s \leq u_s \qquad \forall\, s \in S \tag{2.19}$$

$$\sum_{\pi \in \Omega^k} \theta^k_\pi = 1 \qquad \forall\, k \in K \tag{2.20}$$

$$X^k_{ij} = \sum_{\pi \in \Omega^k} x^k_{ij,\pi} \theta^k_\pi \qquad \forall\, k \in K, \forall\, (i,j) \in A^k \tag{2.21}$$

$$\theta^k_\pi \geq 0 \qquad \forall\, k \in K, \forall\, \pi \in \Omega^k \tag{2.22}$$

$$X^k_{ij} \in \{0,1\} \qquad \forall\, k \in K, \forall\, (i,j) \in A^k \tag{2.23}$$

Dans ce modèle, la constante c^k_π représente le coût d'un chemin $\pi \in \Omega^k$, alors que les coefficients $p^k_{h,\pi}$ et $q^k_{w,\pi}, h \in H, w \in W$ indiquent les contributions de ce chemin dans les contraintes de couverture de tâches (2.17) et dans les contraintes liantes (2.18) respectivement. Les contraintes (2.20) sont des contraintes de convexité associées à la commodité k. Nous notons que nous ajoutons un arc $(o(k), d(k))$ à A^k pour représenter les chemins vides. Sinon, les égalités (2.20) devraient s'écrire comme inégalités, car les commodités ne sont pas

nécessairement toutes utilisées dans la solution.

2.2.2 Sous-problèmes

Les SPs, comme mentionné ci-dessus, sont définis par les domaines δ^k. Chaque SP est principalement un SPPRC. Ce dernier consiste à définir parmi l'ensemble de chemins reliant deux noeuds dans un réseau, celui de coût réduit minimum qui respecte des restrictions imposées par les règles de travail des applications industrielles. Ces restrictions sont modélisées sous forme de contraintes de ressources. Dans ce qui suit, nous décrivons en détails différents aspects requis pour une bonne compréhension du SPPRC.

2.2.2.1 Les contraintes de ressources

On appelle *variable de ressource* toute variable qui sert à modéliser une grandeur telle que le temps de parcours, le poids de produits collectés ou livrés par un véhicule ou bien le temps de travail d'un chauffeur ou de sa durée de repos, etc. Les contraintes de ressources (2.7) sont imposées sur les noeuds du réseau par le biais de $|\mathcal{R}^k|$ fenêtres de ressources $[a_i^{kt}, b_i^{kt}]$ par noeud $i \in V^k$, $t \in \mathcal{R}^k$. Ainsi, pour un réseau $G^k(V^k, A^k)$ associé à la commodité $k \in K$, un sous-chemin de $o(k)$ à un noeud $i \in V^k$ est dit réalisable au noeud i si le cumul de consommations de chaque ressource $R_i^{kt}, t \in \mathcal{R}^k$ le long de ce sous-chemin est contenu dans la fenêtre de ressource correspondante. Un chemin origine destination π est dit réalisable seulement s'il est réalisable dans chacun des noeuds qui le composent. Si on note V_π^k l'ensemble de ces noeuds, la condition de réalisabilité de π s'écrit formellement : $a_i^{kt} \leq R_i^{kt} \leq b_i^{kt}, \forall t \in \mathcal{R}^k, \forall i \in V_\pi^k$.

Il est important de mentionner que cette définition de la réalisabilité en matière de contraintes de ressources peut varier selon la définition du problème à résoudre. En effet, on peut autoriser un certain degré de souplesse des bornes inférieures ou supérieures des fenêtres de temps. Une contrainte est dite souple si on autorise qu'un sous-chemin arrive à un noeud i avec une consommation de ressource en dehors de la fenêtre de temps, en pénalisant les écarts positifs $(a_i^{kt} - R_i^{kt})$ et/ou $(R_i^{kt} - b_i^{kt})$ dans la fonction objectif. Par contre, si la contrainte est rigide, les sous-chemins dont les consommations ne respectent pas les bornes de ressources sont généralement déclarés non réalisables, pourtant, on peut tolérer que $R_i^{kt} < a_i^{kt}$ pour certaines applications.

Le calcul des consommations de ressources cumulées le long d'un chemin est réalisé par le biais de fonctions de prolongation. Une fonction de prolongation est une application : $f_{ij}^{kt} : \mathbb{R}^{|\mathcal{R}^k|} \rightarrow \mathbb{R}$ qui vérifie la mise à jour de la valeur cumulée d'une ressource $t \in \mathcal{R}^k$ suite à chaque prolongation sur un arc $(i, j) \in A^k$. Les problèmes académiques du SPPRC

utilisent généralement des fonctions de prolongation linéaires qui dépendent uniquement de la valeur cumulée de la ressource en question sans tenir compte des valeurs cumulées des autres ressources. Il s'agit de l'application : $f_{ij}^{kt} : \mathbb{R} \to \mathbb{R}$ telle que : $f_{ij}^{kt}(R_i^k, r_{ij}^{kt}) = R_i^{kt} + r_{ij}^{kt} \leq R_j^{kt}$. Cependant, les fonctions de prolongation ne sont pas toujours linéaires, et peuvent différer d'une ressource à l'autre et d'une commodité à l'autre.

2.2.2.2 La fonction objectif

Le lien entre le MP et les SPs s'articule principalement sur la définition de la fonction objectif des SPs. Cette fonction est calculée en prenant en considération les valeurs des variables duales du MP. Soient $\alpha = \{\alpha_h, h \in H\}, \beta = \{\beta_w, w \in W\}$ et $\gamma = \{\gamma_k, k \in K\}$ les vecteurs des valeurs duales associées respectivement aux contraintes (2.17), (2.18) et (2.20). Le coût réduit associé à une variable de chemin $\theta_p^k \in \Omega^k$ est donné comme suit :

$$\begin{aligned} c_\pi^k(\alpha, \beta, \gamma) &= c_\pi^k - \sum_{h \in H} p_{h,\pi}^k \alpha_h - \sum_{w \in W} q_{w,\pi}^k \beta_w - \gamma_k \\ &= c_\pi^k - \sum_{(i,j) \in A^k} (\sum_{h \in H} p_{h,ij}^k \alpha_h + \sum_{w \in W} q_{w,ij}^k \beta_w) x_{ij,\pi}^k - \sum_{i \in V^k} \sum_{t \in \mathcal{R}^k} (\sum_{w \in W} q_{w,ij}^k \beta_w) \tau_{i,\pi}^{kt} - \gamma_k \end{aligned}$$

En termes de variables de flot sur les arcs et celles de ressources dans le modèle original, la fonction objectif du SP k s'écrit :

$$Obj_{SP}^k = c^k - \sum_{(i,j) \in A^k} (\sum_{h \in H} p_{h,ij}^k \alpha_H + \sum_{w \in W} q_{w,ij}^k \beta_w) X_{ij}^k - \sum_{i \in V^k} \sum_{t \in \mathcal{R}^k} (\sum_{w \in W} q_{w,ij}^k \beta_w) R_i^{kt} - \gamma_k$$

Où : $c^k = \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k X_{ij}^k + \sum_{k \in K} \sum_{i \in V^k} \sum_{t \in \mathcal{R}^k} c_i^{kt} R_i^{kt}$

Afin d'alléger la notation, nous omettons dans ce qui suit l'indice de la commodité k. Ceci donne naissance à la formulation suivante du SP qui utilise les variables du modèle original :

$$Min \quad Obj_{SP} \tag{2.24}$$

sous les contraintes :

$$\sum_{j:(i,j) \in A} X_{ij} - \sum_{j:(j,i) \in A} X_{ji} = \begin{cases} -1 & \text{for } i = o \\ 0 & \forall\ i \in N \\ 1 & \text{for } i = d \end{cases} \tag{2.25}$$

$$X_{ij}(f_{ij}^t(R_i, r_{ij}^t) - R_j^t) \leq 0 \qquad \forall\, t \in \mathcal{R}, \forall\, (i,j) \in A \tag{2.26}$$

$$a_i^t\,(\sum_{j \in V} X_{ij}) \leq R_i^t \leq b_i^t\,(\sum_{j \in V} X_{ij}) \qquad \forall\, t \in \mathcal{R}, \forall\, i \in V \tag{2.27}$$

$$X_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{2.28}$$

Les SPs servent alors à construire des chemins réalisables de coûts réduits négatifs qui peuvent améliorer la solution du MP. C'est l'idée fondamentale de la méthode de CG.

2.2.3 Méthode de génération de colonnes

L'idée centrale de la méthode de CG dérive principalement de la décomposition de Dantzig & Wolfe (1960). Pourtant, le MP exige la connaissance de tous les points extrêmes $\{\Omega^k, k \in K\}$, ce qui est souvent très loin de la portée des décideurs en pratique. Et même si ces points extrêmes sont tous connus à l'avance, ils donnent naissance à des problèmes de taille énormément grande dont la résolution est impossible. Dans la plupart de ces problèmes, on constate qu'une grande portion des variables (colonnes) associées à ces points extrêmes demeurent hors de la base dans la solution optimale et seul un petit sous-ensemble de ces variables contribue effectivement à la résolution du problème.

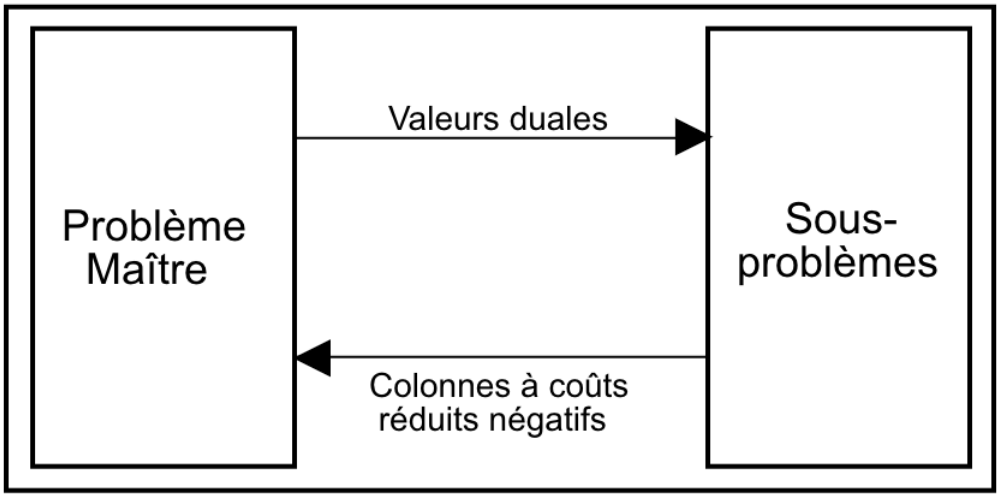

Figure 2.2 Algorithme de CG

La méthode de CG initialise le MP avec un sous-ensemble de variables de taille réduite, ce qui donne naissance à un RMP relativement facile à résoudre. Le mécanisme de la méthode consiste alors à enrichir ce problème avec les colonnes les plus susceptibles d'améliorer la solution courante du problème, i.e., les colonnes de coûts réduits négatifs dans le cas de

minimisation. La génération de ces colonnes est faite de façon itérative par le biais de la résolution des SPs. Le coût réduit d'une nouvelle colonne générée par un SP donné est tout simplement le coût réduit du chemin correspondant.

En résumé, la méthode de CG est un processus itératif qui résout alternativement un RMP et un ou plusieurs SPs. Le processus de résolution continue tant que les SPs sont capables de produire des colonnes de coûts réduits négatifs et s'arrête immédiatement lorsque cette condition ne tient plus. Il s'agit de la fameuse condition d'optimalité de l'algorithme du simplexe pour résoudre les problèmes linéaires. Le schéma 2.2 illustre le fonctionnement de la méthode.

CHAPITRE 3 REVUE DE LITTÉRATURE

Pour la plupart des applications, les problèmes de tournées de véhicules et d'horaires d'équipages sont résolus à l'aide d'un algorithme de CG dans lequel les SPs correspondent à des instances de SPPRC ou de l'une de ses variantes. Le SPPRC est apparu pour la première fois dans la dissertation de doctorat de Desrochers (1986) comme SP lors de la résolution par CG du problème d'horaires de chauffeurs de bus.

Dans ce chapitre, nous décrivons dans un premier lieu les différentes variantes du SPPRC ainsi que sa complexité. Ensuite, nous examinons en détail les différentes méthodes de résolution qui ont été développées dans la littérature pour chaque variante du problème. Finalement, nous citons quelques domaines d'application du SPPRC.

3.1 Variantes et complexité du SPPRC

Vu la flexibilité de la formulation du SPPRC et sa capacité de s'adapter à différentes situations industrielles, plusieurs variantes et extensions de ce problème sont apparues dans la littérature. Ces variantes se distinguent principalement selon deux paramètres majeurs :

1. les ressources, à savoir leur nature, leur nombre et la nature des fonctions de prolongation qui leur sont associées.
2. les réseaux de base et les restrictions sur les structures de chemins que ces derniers peuvent engendrer.

En ce qui concerne les ressources considérées, la littérature cite tout d'abord le problème de plus court chemin contraint (Constrained Shortest Path Problem - CSPP), ce dernier impose une seule contrainte de borne supérieure (ou inférieure) au nœud destination (voir Dumitrescu & Boland, 2003). Une variante de ce problème considère plusieurs contraintes de borne supérieure (ou inférieure) au nœud destination au lieu d'une seule (Resource Constrained Shortest Path Problem - RCSPP)(voir Pugliese & Guerriero, 2013a). Le problème qui considère une seule ressource imposant une borne inférieure et supérieure en chaque nœud du réseau est connu sous le nom de plus court chemin avec fenêtres de temps (Shortest Path Problem with Time Windows - SPPTW)(voir Desaulniers & Villeneuve, 2000). Une généralisation de ce problème donne naissance au problème de plus court chemin avec contraintes de ressources (Shortest Path Problem with Resource Constraints - SPPRC), cette version considère plusieurs contraintes de ressources de type fenêtre de temps au lieu d'une seule (voir Nagih & Soumis, 2006). Il est important de mentionner que le SPPRC généralise égale-

ment le RCSPP, car les contraintes de bornes supérieures (ou inférieures) ne sont qu'un cas particulier des contraintes de fenêtre de temps.

Dans le cas de réseaux cycliques, la littérature définit pour chaque classe de problème une version élémentaire appropriée. Un chemin est dit *élémentaire* s'il ne passe pas plus qu'une fois par un même nœud. La condition d'élémentarité de chemins n'est pas définie lorsque les réseaux considérés sont acycliques. D'autres restrictions sur la structure des chemins générés ont donné naissance au problème de plus court chemin avec chemins interdits (Shortest Path Problem with Forbidden Paths - SPPFP), ce dernier consiste à définir un chemin de coût minimal tout en empêchant des séquences d'arcs de faire partie d'une solution réalisable.

Au niveau de la complexité, Garey & Johnson (1979), Handler & Zang (1980) et Jaffe (1984) ont prouvé que l'ajout de contraintes au problème classique de plus court chemin rend ce dernier NP-difficile. Un résultat que Dumitrescu & Boland (2003) ont affirmé même dans le cas d'une seule contrainte de ressource et sur un réseau acyclique avec coût et ressources positifs. La difficulté du problème augmente très rapidement en fonction du nombre de ressources.

3.2 Méthodes de résolution

Dans ce qui suit, nous présentons une revue des méthodes de résolution qui ont marqué l'histoire des recherches effectuées en vue de résoudre les différentes variantes du SPPRC. Ces travaux de recherche se sont concentrés essentiellement sur le CSPP, son extension à plusieurs ressources RCSPP et aussi sur le SPPTW (Nagih & Soumis, 2006). Cependant, les travaux s'adressant au problème le plus général du SPPRC n'ont pas reçu le même niveau d'intérêt.

La revue de littérature que nous proposons est présentée par classe de problèmes. Nous nous intéressons surtout aux méthodes exactes, pourtant nous nous permettons de citer pour chaque classe quelques approches heuristiques les plus marquantes.

3.2.1 Problème de plus court chemin contraint : CSPP et RCSPP

Le problème de plus court chemin avec contraintes de capacité consiste à trouver un chemin de coût minimal tel que le cumul de consommation de chacune des ressources considérées est au plus égal à une borne supérieure au nœud destination.

Pugliese & Guerriero (2013b) ont établi un cadre général permettant de classifier les différentes contributions développées dans la littérature selon le stade d'intervention. Selon les deux auteurs, ces contributions touchent une ou plusieurs des 3 principales étapes suivantes :

1. une étape de prétraitement qui vise à réduire la taille du réseau, 2. une étape de calcul des bornes inférieures et supérieures pour réduire la taille de l'espace de recherche, 3. une étape de réduction de l'écart, ce qui permet de retourner des solutions optimales.

3.2.1.1 Prétraitement : Réduction du réseau

Les techniques de prétraitement visent à éliminer les nœuds et les arcs qui n'ont pas de chance de faire partie d'un chemin réalisable optimal. Aneja *et al.* (1983) furent les premiers à utiliser ces techniques pour résoudre le CSPP. Considérons un réseau $G(V, A)$ de noeud source s et de noeud destination d et un ensemble de ressources $\mathcal{R}$. Notons par π^c_{ij} et π^t_{ij}, $t \in \mathcal{R}$ les chemins les plus courts entre deux noeuds i et j en termes de coût et de consommation de la ressource respectivement. Soient $C(\pi^c_{ij})$ et $R(\pi^t_{ij})$, $t \in \mathcal{R}$ les valeurs respectives de coût et de consommations de ressources associées à ces chemins. Si B^t est une borne supérieure de la ressource $t \in \mathcal{R}$ au nœud destination, tout nœud $i \in V$ tel que $R(\pi^t_{si}) + R(\pi^t_{id}) > B^t$ ne pourrait jamais faire partie d'un chemin réalisable, un tel nœud peut être retiré du réseau en toute sécurité. De même, un arc $(i, j) \in A$ est éliminé si $R(\pi^t_{si}) + r^t_{ij} + R(\pi^t_{jd}) > B^t$, où r^t_{ij} est la consommation de la $t^{ème}$ ressource sur l'arc (i, j). Le même raisonnement permet de réduire la taille du réseau pour le cas des ressources de borne inférieure, il suffit dans ce cas de calculer le plus long chemin au lieu du plus court chemin.

Ces règles d'élimination ont été appliquées dans un algorithme itératif proposé par Dumitrescu & Boland (2003). Ces derniers ont tiré profit de la qualité des bornes sur le coût, ce qui leur a permis d'énormes réductions au niveau de la taille des réseaux considérés. Beasley & Christofides (1989) puis Melhorn & Ziegelmann (2000) ont proposé, en plus de l'élimination par consommation de ressources, de nouvelles règles d'élimination basées sur les coûts réduits des arcs. Ces derniers sont calculés à l'aide des valeurs duales qui dérivent de la résolution du problème dual Lagrangien. Une contribution récente de Carlyle *et al.* (2008) propose d'appliquer les règles d'Aneja *et al.* (1983) pour une ressource agrégée, au lieu de les appliquer séparément sur chaque ressource. L'efficacité de leur méthode de réduction a été illustrée à l'aide d'un exemple numérique.

3.2.1.2 Calcul des bornes

La relaxation Lagrangienne demeure la méthode la plus utilisée pour obtenir des bornes inférieures au RCSPP. Cette méthode consiste tout d'abord à résoudre une version simplifiée du problème appelée *problème dual Lagrangien*, puis elle essaie de réduire l'écart de dualité en utilisant différentes stratégies. Cette alternative a été largement étudiée dans la littérature.

En 1980, Handler & Zang (1980) ont proposé une méthode pour résoudre le problème dual Lagrangien du CSPP en faisant appel à la méthode des plans coupants de Kelley Jr. (1960). Pour réduire l'écart/gap de dualité, ils ont appliqué l'algorithme de k-plus courts chemins de Yen (1971) en affectant aux arcs les coûts réduits obtenus en utilisant les multiplicateurs de Lagrange associés à la solution optimale. Dumitrescu & Boland (2003) ont proposé une approche similaire à celle de Handler et Zang pour résoudre le problème dual Lagrangien. Pourtant, ils ont introduit une technique qui sauvegarde les multiplicateurs associés à toutes les solutions intermédiaires pour s'en servir afin de renforcer la réduction de l'écart/gap de dualité.

Beasley & Christofides (1989) ont utilisé la méthode du sous-gradient lors de la résolution du problème dual Lagrangien. De leur part, Carlyle *et al.* (2008) ont publié une approche similaire à celle de Handler & Zang (1980), sauf que les multiplicateurs de Lagrange sont définis à l'aide d'une méthode de bissection, alors que la réduction du gap de dualité se fait avec la méthode de Branch & Bound. Melhorn & Ziegelmann (2000) ont développé un nouvel algorithme qui résout le problème dual correspondant à la relaxation linéaire du RCSPP à l'aide de la méthode de l'Ellipsoïde. Les auteurs ont également associé une interprétation géométrique à cet algorithme avant de prouver sa polynomialité pour le cas d'une seule contrainte de ressource sous certaines hypothèses.

Très récemment, Pugliese & Guerriero (2013a) ont formulé le RCSPP en utilisant une approche basée sur le concept de point de référence. Ce point est modifié au cours de la résolution en utilisant une direction de recherche. Cette dernière est définie en se servant de l'information qui provient de la dernière solution réalisable obtenue. La mise à jour des bornes inférieures et supérieures est effectuée à l'aide d'un algorithme d'étiquetage. Dans la plupart des cas, les bornes inférieures obtenues constituent des solutions optimales des instances considérées.

3.2.1.3 Réduction de l'écart (gap)

Plusieurs méthodes ont été proposées pour réduire l'écart de dualité obtenu par la résolution de la relaxation. Ces méthodes ont été classifié par Pugliese & Guerriero (2013b) selon le principe de résolution en trois classes principales :

1. Méthodes basées sur le branchement (*Branching methods*).
2. Méthodes de classement de chemins (*Path Ranking* or *k-Shortest Path methods*).
3. Méthodes d'étiquetage ou de programmation dynamique (*Labeling* or *Dynamic Programming methods*).

Il est important de noter que les méthodes faisant partie des deux dernières classes peuvent

être utilisées directement pour trouver une solution optimale du SPPRC sans même avoir à résoudre le problème relaxé. Par la suite, nous présentons un aperçu des contributions de chacune des trois classes mentionnées ci-dessus.

3.2.1.3.1 Méthodes de branchement

Les méthodes de branchement créent un arbre de branchement dans lequel chaque nœud correspond à un sous-chemin de la source s à un nœud i du réseau. Les nœuds associés à des sous-chemins qui n'ont pas de chance de produire des chemins réalisables sont élagués de l'arbre. Dans ce cas, un retour en arrière est effectué pour explorer d'autres branches. Le choix des branches à parcourir est régi par des règles de branchement. Les algorithmes faisant appel à cette classe de méthodes calculent continuellement des bornes supérieures et inférieures afin de réduire la taille du problème autant que possible. La qualité de ces bornes affecte amplement l'efficacité de ces méthodes. Parmi les contributions s'adressant à des méthodes de branchement, nous citons dans l'ordre de parution Beasley & Christofides (1989), Muhandiramge & Boland (2009) et Carlyle *et al.* (2008).

3.2.1.3.2 Méthodes de classement de chemins

Les méthodes de classement de chemins consistent à relâcher complètement les contraintes de ressources avant de construire une séquence de chemins dans un ordre croissant de coût. Un chemin précédemment généré est interdit d'être reproduit dans une itération ultérieure, et ce en utilisant une variété de techniques d'élimination de chemins. Ce processus continue jusqu'à ce qu'un chemin, soit le $(k+1)^{\text{ème}}$, soit réalisable. Ce chemin est nécessairement une solution optimale du problème contraint. L'idée originale de cette méthode revient à Handler & Zang (1980) qui étaient les premiers à extraire les k-plus courts chemins en se fondant sur les coûts réduits des arcs calculés à l'aide des multiplicateurs de Lagrange.

Cette méthode présente malheureusement l'inconvénient que le nombre de chemins que l'on devrait extraire risque d'être très grand, ce qui alourdit le processus de résolution. Pour remédier à ce problème, Santos *et al.* (2007) ont développé un nouvel algorithme permettant de définir les k-plus courts chemins de façon plus puissante par rapport à l'approche initiée par Handler & Zang (1980). L'algorithme proposé consiste à classifier les chemins en utilisant une combinaison convexe du coût et des valeurs de consommations de ressources. Récemment, Pugliese & Guerriero (2013a) ont introduit une nouvelle approche faisant appel à une norme de type $Tchebychev$ pour évaluer la qualité des chemins. Ils font varier un point de référence afin de produire des chemins différents, une procédure qui ne cesse de s'améliorer jusqu'à

obtention d'une solution optimale.

3.2.1.3.3 Méthodes de programmation dynamique

Soit $G(V, A)$ un réseau, et notons $\mathcal{R}$ un ensemble de ressources. Les approches de DP, également appelés *algorithmes d'étiquetage*, consistent à construire graduellement des sous-chemins à partir du nœud source s. Ces algorithmes affectent à chaque nœud i du réseau un ensemble d'états dont chacun est associé à un sous-chemin allant de l'origine au nœud i. Chaque état est caractérisé par un vecteur $l = (C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|})$ appelé *étiquette* qui mémorise le coût C_l et les consommations cumulées de ressources R_l^t, $t \in \mathcal{R}$ le long du sous-chemin correspondant.

Un algorithme d'étiquetage fonctionne en deux étapes : une étape de prolongation et une étape de dominance. La prolongation permet d'étendre les sous-chemins d'un nœud donné vers ses successeurs. Cette extension se fait à l'aide d'une fonction de prolongation qui assure la bonne mise à jour du coût et des valeurs de consommations de ressources. Cette fonction vérifie également la réalisabilité des étiquettes, et élimine celles correspondant à des sous-chemins non-réalisables.

La dominance permet de comparer chaque couple de sous-chemins arrivant au même nœud, afin d'arrêter l'extension des sous-chemins dominés. Un sous-chemin est dit *dominé* s'il ne peut jamais conduire à une solution optimale. Une telle affirmation est intimement liée aux règles de dominance considérées. Plusieurs règles de dominance apparaissent dans la littérature. Dans le cas où les fonctions de prolongation sont non-décroissantes, la règle de dominance la plus utilisée en pratique est définie comme suit :

Definition 1. *Soient $l = (C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|})$ et $p = (C_p, R_p^1, R_p^2, ..., R_p^{|\mathcal{R}|})$ deux étiquettes associées respectivement à deux sous-chemins π_l et π_p dans un noeud $i \in V$. L'étiquette l est dite dominée par p si $C_p \leq C_l$ et $R_p^t \leq R_l^t$, $\forall t \in \mathcal{R}$ tel qu'au moins une de ces inégalités est stricte. Le sous-chemin π_l est dit dominé, alors que π_p est dit dominant.*

L'efficacité de ce genre d'algorithmes dépend de leur capacité d'identifier les étiquettes correspondant à des sous-chemins non-réalisables ou dominés. En effet, l'élimination de ces étiquettes réduit énormément la taille de l'espace des états, et permet d'accélérer le processus de résolution.

La première utilisation d'un algorithme de DP en vue de résoudre un problème de plus court chemin contraint revient à Aneja *et al.* (1983). Il s'agit d'une généralisation de l'algorithme de Dijkstra. Un autre algorithme de DP a été introduit par Melhorn & Ziegelmann (2000). Ce

dernier résout un problème dual Lagrangien et utilise l'information duale qui dérive de cette résolution afin de calculer des coûts réduits des étiquettes. Ces dernières sont alors traitées selon leurs coûts réduits au sein d'un algorithme de DP. Dumitrescu & Boland (2003) ont proposé une approche similaire qui tire profit des multiplicateurs de Lagrange. Cependant, contrairement à Melhorn & Ziegelmann (2000), les étiquettes sont traitées dans un ordre croissant des valeurs des consommations de ressources.

D'autres travaux de recherche se sont intéressés à l'aspect multicritère du problème. White (1982) a été le premier à combiner la DP et la programmation linéaire en transformant les contraintes de ressources en objectifs du SPPRC. De sa part, Heing (1986) a utilisé une approche multicritère qui intègre la DP dans un algorithme de plus court chemin afin de définir des chemins non dominés ou Pareto-optimaux.

3.2.2 Problème de plus court chemin avec fenêtres de temps : SPPTW

Le SPPTW est apparu pour la première fois dans un article de Desrosiers *et al.* (1983) sous le nom de *plus court chemin avec contraintes d'horaires*. Les algorithmes proposés en vue de résoudre le SPPTW sont tous de type DP (Pugliese & Guerriero, 2013b). Pourtant, ces algorithmes diffèrent selon la façon utilisée pour traiter les étiquettes, cette distinction permet de classifier ces algorithmes en deux classes : les algorithmes de type *label-setting* et ceux de type *label-correcting*. Les deux classes d'algorithmes sont itératives et font appel à une fonction de prolongation afin d'explorer l'espace des états.

Une spécificité des algorithmes de type *label-setting* est qu'une étiquette réalisable dans un nœud donné n'est prolongée que lorsqu'il est certain qu'elle est définitive et qu'elle ne peut en aucun cas être dominée au cours des itérations subséquentes. Par contre, les étiquettes traitées par un algorithme de type *label-correcting* sont temporaires et peuvent être prolongées comme elle peuvent être éliminées au cours des itérations. Une comparaison détaillée de ces deux classes d'algorithmes est offerte par Zhan (2000).

Un premier algorithme de type *label-correcting* a été développé pour le SPPTW par Desrosiers *et al.* (1983). Il s'agit d'une généralisation du fameux algorithme de Bellman-Ford, qui considère dans chaque nœud un ensemble d'étiquettes sous forme de couple *(temps, longueur)*. L'algorithme fournit un ensemble de chemins optimaux au sens de Pareto, dans un temps exponentiel au pire cas. Powell & Chen (1998) ont proposé un autre algorithme de la même classe qui est directement applicable à un problème à plusieurs ressources. Cet algorithme manipule trois listes d'étiquettes triées selon un ordre lexicographique et utilise une étiquette seuil pour gérer les déplacements des étiquettes entre les trois listes.

Desrochers & Soumis (1988a) ont été les premiers à s'adresser au SPPTW avec un algorithme de type *label-setting*. C'est un algorithme pseudo-polynomial qui utilise un concept de *bucket* (ou seau) d'étiquettes pour établir un ordre de traitement des étiquettes. La construction de ces *buckets* d'étiquettes est basée sur un ordre lexicographique. Un autre *label-setting* algorithme est introduit par Ioachim *et al.* (1998), pour une version modifiée du SPPTW. D'abord, cet algorithme établit un tri topologique des nœuds grâce à l'acyclicité des réseaux considérés. Ensuite, il prend en considération des coûts additionnels associés au temps d'arrivée à chaque nœud.

3.2.3 Problème de plus court chemin avec contraintes de ressources : SPPRC

La première généralisation du SPPTW au cas de plusieurs ressources de type fenêtres de temps a été réalisée par Desrochers (1986) dans sa thèse de doctorat. Dès lors, plusieurs travaux ont essayé d'adapter les méthodes de résolution développées pour le SPPTW au cas du SPPRC. Une synthèse détaillée du SPPRC, sa formulation, ses variantes et ses méthodes de résolution est présentée par Irnich & Desaulniers (2005).

L'approche standard pour résoudre le SPPRC en pratique est basée sur la DP. Cette approche a été développée par Desrochers *et al.* (1992) suite à un ajustement de l'algorithme de type *label-setting* proposé par Desrochers & Soumis (1988a) pour le SPPTW. Une adaptation de cette approche a été introduite par Feillet *et al.* (2004) pour un SPPRC élémentaire. L'approche consiste à introduire de nouvelles ressources afin d'interdire la génération de chemins non-élémentaires. Dans le même souci, Irnich & Villeneuve (2006) ont proposé un nouvel algorithme de type DP qui permet d'omettre les cycles de longueurs ≥ 3.

Cette classe de méthodes devient relativement fastidieuse lorsque le nombre de ressources considérés est grand. En effet, les chances d'élimination de sous-chemins par dominance deviennent limitées, ce qui augmente rapidement l'espace des états. Par conséquent, la gestion des étiquettes devient drastiquement coûteuse aussi bien en matière de mémoire de stockage qu'en matière de temps d'exécution. Pour remédier à ce problème, plusieurs recherches ont été dédiées à l'étude de la dominance afin de réduire l'espace des états. Dans cette optique, Nagih & Soumis (2006) ont proposé de projeter les vecteurs de ressources sur un espace de dimension inférieure au nombre de ressources. Cette projection leur a permis de construire de nouvelles ressources agrégées qui ont été utilisées par les règles de dominance. Cette idée a permis de réduire énormément le nombre d'étiquettes par nœud, cependant, elle a causé une perte des étiquettes optimales. Afin d'alléger cette perte, les auteurs ont proposé un ajustement dynamique des cœfficients de la matrice de projection à l'aide des multiplicateurs de Lagrange. Ce remède a permis de donner une meilleure approximation de la solution

optimale.

3.3 Applications du SPPRC

Le SPPRC apparaît généralement comme SP lors de la résolution de problèmes de grande taille avec une méthode de CG. Ces problèmes ont une structure particulière qui admet une décomposition de Dantzig & Wolfe (1960) comme mentionné dans la section 2.2. Cette structure apparaît de façon redondante dans diverses applications dont une grande partie relève principalement du domaine du transport.

Le SPPRC a été utilisé en transport urbain lors de la résolution du problème de blocs mensuels de chauffeurs de bus (Desrochers & Soumis, 1989). Le problème s'applique aussi en transport scolaire (Desrosiers *et al.*, 1984) et en transport de marchandises (Desrochers *et al.*, 1992) qui s'énoncent comme problèmes de tournée de véhicules avec fenêtres de temps. Le SPPRC a servi également à optimiser la qualité des services de transport des personnes handicapées (Desrosiers *et al.*, 2003; Ioachim *et al.*, 1995). En transport ferroviaire, nous rappelons le travail de Ziarati *et al.* (1997). En transport aérien, la croissance continue de l'utilisation du transport aérien ainsi que la complexité des règles de gestion ont donné naissance à des problèmes difficiles de taille énorme. Pour cette raison, le SPPRC s'impose de façon inévitable lors de la résolution avec CG de ces problèmes. Parmi les applications aériennes du SPPRC, nous citons le problème d'horaires d'équipages (Desrochers & Soumis, 1989; Mingozzi *et al.*, 1999) et le problème de routage des avions long-courriers (Barnhart *et al.*, 1998).

À part le transport, le SPPRC a été également utilisé pour résoudre une grande variété de problèmes d'optimisation avec CG. Entre autres, nous rapportons le problème de prescription du contenu et du calendrier des mises à niveau de produits (Wilhelm *et al.*, 2003), le problème d'optimisation des opérations de prélèvement (Wilhelm *et al.*, 2006) et de placement sur les machines de placement à deux têtes (Wilhelm *et al.*, 2007) et le problème de flux à multi-commodité (Holmberg & Yuan, 2003).

En plus de son utilité dans un cadre de CG, la formulation du SPPRC a permis également de modéliser différentes situations où l'objectif est de minimiser le coût de certaines opérations tout en vérifiant un ensemble de contraintes, et ce dans divers domaines d'application. En transport, nous rappelons, à titre d'exemples, le travail de Halpern & Priess (1974) pour une gestion optimale des routes ferroviaires et celui de Zabarankin *et al.* (2001) qui ont proposé une application aux systèmes de gestion d'aéronefs militaires. En outre, le SPPRC a été utilisé en abondance dans le domaine de la télécommunication, notamment pour le problème de conception des réseaux de télécommunication (Cabral, 2005).

CHAPITRE 4 PROBLÉMATIQUE ET CONTRIBUTIONS

Dans ce chapitre, nous exposons la problématique adressée par cette thèse. Ensuite, nous décrivons les grandes lignes des différentes contributions et améliorations que nous proposons pour remédier à cette problématique.

4.1 Description de la problématique

Il est connu que l'efficacité de la méthode de CG dépend largement du mécanisme utilisé pour résoudre les SPs. La résolution de ces problèmes est souvent considérée en pratique comme étant une tâche compliquée et drastiquement coûteuse même pour les solveurs les plus sophistiqués. D'un côté, au cours de la résolution par CG de problèmes d'optimisation liés à quelques applications aériennes, on devrait résoudre à chaque itération des centaines de SPs (un SP par pilote ou par avion à titre d'exemple). Le nombre d'itérations de CG pour cette classe de problèmes est de l'ordre d'un millier. Il s'agit par conséquent de résoudre des centaines de milliers de SPPRC.

D'un autre côté, pour chaque problème, on doit s'occuper de plusieurs contraintes de ressources qui reflètent les règles complexes de travail. À titre d'exemple, le problème de blocs mensuels de chauffeurs (Desrochers, 1986) impose un nombre de ressources allant de trois à cinq ; ce nombre varie entre dix et vingt pour le problème de blocs mensuels d'équipages de vols (Desaulniers *et al.*, 1998a). En outre, ces problèmes sont souvent définis sur des réseaux composés de dizaines de milliers de nœuds et de centaines de milliers d'arcs.

Si on prend en considération que les SPPRCs sont classés NP-difficile même lorsqu'il s'agit d'une seule contrainte de ressource (Handler & Zang, 1980), la difficulté du problème est par conséquent multidimensionnelle. En fait, on fait face à un problème NP-difficile qui devrait être résolu sur des instances de grande taille, plusieurs fois par itération, et ce au sein d'un processus faisant appel à un grand nombre d'itérations.

Les solveurs commerciaux utilisant la méthode de CG font appel à la méthode de DP (voir chapitre 3) pour résoudre les SPs à chaque itération. Cette méthode s'avère la plus adéquate au contexte de CG pour trois fortes raisons : 1. elle est capable de donner des solutions entières, c-à-d : des chemins réalisables, ce qui élimine toute difficulté reliée à l'intégralité de la solution, 2. elle peut générer plusieurs colonnes à chaque itération au lieu d'une seule, comme c'est le cas pour la relaxation Lagrangienne par exemple, 3. elle peut traiter des problèmes complexes qui peuvent même être non linéaires ou non convexes. Toutefois, la DP

souffre de deux inconvénients majeurs :

1. La première faiblesse de la DP est due principalement au fameux problème de la dimensionnalité. En pratique, lorsque le nombre de ressources dépasse quatre à cinq et que ces ressources sont faiblement corrélées, les règles de dominance perdent leur efficacité. Ainsi, un nombre exponentiel d'étiquettes non prometteuses sont conservées. Par conséquent, l'espace d'états grandit rapidement surtout pour des réseaux de grande taille. La mémorisation ainsi que la manipulation de ces étiquettes deviennent dans ce cas très coûteuses à la fois en matière d'espace de stockage et en matière de temps de calcul.
2. Le deuxième inconvénient provient des besoins de la CG face à l'aspect dual de la DP. En effet, cette méthode ne permet généralement pas d'avoir des chemins réalisables au cours de la résolution. Le traitement des étiquettes est fait en une seule étape, et les solutions réalisables ne sont retournées qu'à la fin de l'algorithme lorsque toutes les extensions possibles de sous-chemins sont faites. Cependant, nous rappelons que l'utilité des SPs dans un cadre de CG est simplement de nourrir le RMP par des chemins réalisables de coûts réduits négatifs. Ce principe permet de voir facilement que l'optimalité des solutions des SPs n'est requise qu'à la fin du processus afin de vérifier la condition d'arrêt de l'algorithme. Toutefois, il n'est pas exigé pour la majorité des itérations de la CG de trouver une solution optimale. Cette particularité rend la méthode de DP incompatible avec les exigences de la CG.

Différentes approches heuristiques ont été proposées pour corriger les faiblesses de la DP. Les solutions les plus appliquées en pratique consistent à réduire la complexité du problème en limitant la taille de l'espace des états. Une première stratégie propose de définir une borne supérieure sur le nombre d'étiquettes qu'un nœud pourrait contenir. Le choix des étiquettes à conserver dans un nœud parmi l'ensemble d'étiquettes arrivant au même nœud se fait généralement selon le critère des coûts réduits des étiquettes. La deuxième approche qui est le plus souvent adoptée afin de contourner cette difficulté consiste à dominer sur un sous-ensemble de ressources (ressources dominantes). Cette méthode est statique au sein de la même résolution. Pourtant, elle permet au décideur de modifier les sous-ensembles de ressources sélectionnées au cours des itérations, afin de varier la qualité des colonnes générées.

Les solutions proposées, bien qu'elles soient capables d'alléger le réseau, accélérer la résolution et fournir des chemins réalisables de bonne qualité à moindre effort, restent loin de subvenir aux besoins de la CG. Cette affirmation est due à plusieurs raisons. Premièrement, le choix des étiquettes à garder ainsi que la sélection des ressources dominantes sont plutôt basés sur l'intuition du programmeur-analyste que sur des règles mathématiquement fondées.

Théoriquement, ces stratégies n'offrent par conséquent aucune garantie sur la qualité de la solution et l'algorithme risque facilement de tomber sous-optimal. Deuxièmement, les améliorations proposées n'envisagent aucun apprentissage des résolutions passées en cas d'échec de ces dernières. En effet, si par exemple la dominance heuristique ne réussit pas à générer des solutions réalisables de coûts réduits négatifs, l'algorithme devrait recommencer la résolution du SP à zéro (from scratch). Dans ce cas, il faut élargir le sous-ensemble de ressources ou bien en considérer un nouveau sous-ensemble entièrement différent. Troisièmement, ces méthodes heuristiques exigent une connaissance préalable des problèmes à résoudre, un ajustement par itération, en plus d'un suivi et d'une validation réguliers de la part du modeleur. Ces aspects sont souvent difficiles à vérifier en pratique. Finalement, l'ajustement a priori des paramètres ne permet pas d'arrêter la résolution des SPs quand les solutions retournées sont jugées suffisamment bonnes. La raison est que la qualité de la solution n'est pas décidable à l'avance.

4.2 Contributions et améliorations proposées

Notre travail s'inscrit dans une optique qui vise à faire face aux faiblesses de la DP lors de la résolution des SPs au sein d'un algorithme de CG. Pour ce faire, nous nous sommes fixés trois objectifs principaux :

1. Concevoir des méthodes itératives exactes permettant une réduction dynamique de la dimension de l'espace des états.
2. Développer des techniques d'apprentissage qui favorisent la résolution en tirant profit des résultats des itérations passées.
3. Établir un cadre qui permet aux algorithmes de CG d'arrêter la résolution des SPs une fois que des solutions satisfaisantes sont trouvées.

Afin de mettre en œuvre les objectifs fixés, nous proposons d'aborder cette problématique en faisant appel à des méthodes primales exactes. Une méthode de résolution est dite *primale*, si elle effectue sa recherche dans le domaine des solutions réalisables. Nous estimons que ce cadre est fortement adapté au SPPRC pour différentes raisons. Premièrement, la plupart de ces méthodes n'exigent pas aux problèmes à résoudre d'avoir des structures spécifiques telles que la convexité. C'est bien le cas pour le SPPRC vu que les contraintes de ressources ne sont généralement pas convexes. Deuxièmement, si le processus de résolution s'arrête avant d'atteindre une solution optimale, le point courant est réalisable. Cette propriété est très intéressante dans un contexte de CG. Troisièmement, contrairement aux méthodes heuristiques, les méthodes primales sont capables d'offrir la garantie qu'un optimum local ou même

global est atteint si la séquence de points considérés converge. Le risque de perdre l'optimalité pourrait donc être facilement contrôlé avec des méthodes primales.

La première contribution de cette thèse est une approche multi-directionnelle qui généralise la recherche unidirectionnelle des méthodes de DP. Il s'agit d'une méthode primale exacte appelée : *Multi-Directional Dynamic Programming Algorithm* (MDDPA). Cette méthode effectue des recherches séquentielles sur des sous-espaces de taille énormément réduite par rapport à l'espace des états induit par le domaine réalisable. Cette méthode de recherche itérative permet de générer des ensembles de chemins réalisables de coûts réduits négatifs non croissants conduisant à l'optimalité. Elle intègre trois idées qui fonctionnent ensemble : 1. *Label Storing Procedure* qui permet de partitionner l'espace des états en plusieurs sous-espaces disjoints de petites tailles. Cette partition est rendue possible grâce à une nouvelle formulation du SPPRC que nous introduisons pour la première fois, 2. *Label Loading Strategies*, on propose deux stratégies différentes qui permettent l'exploration itérative des sous-espaces de recherche, 3. *Learning from the locally efficient labels*. Cette contribution montre dans quelle mesure les résultats des itérations passées et surtout les étiquettes efficaces précédemment générées peuvent contribuer à l'amélioration des itérations subséquentes. L'efficacité du MDDPA est évaluée en comparaison avec un algorithme standard de DP sur des instances de SPPRC extraites de différentes itérations lors de la résolution par CG des instances de VCSP.

Comme deuxième contribution, nous proposons un nouvel algorithme appelé *Primal Adjacency-Based algorithm* (PAB). Nous effectuons en premier lieu une étude polyédrique du problème de plus court chemin, grâce à elle, nous définissons une nouvelle partition de l'espace des solutions en plusieurs sous-espaces disjoints. Chaque sous-espace correspond à un degré d'adjacence dans le polyèdre du plus court chemin par rapport à un point initial formé par un ensemble de chemins initiaux. Ce point initial est construit en tirant profit de l'information primale qui est souvent disponible a priori pour les problèmes industriels. Cette information provient de la structure du problème comme elle peut être déduite des anciennes planifications. L'algorithme PAB explore graduellement les sous-espaces relatifs aux différents degrés d'adjacence. Ainsi, il génère une suite d'ensembles de chemins réalisables tout en garantissant une décroissance de la fonction du coût réduit. En outre, la méthode proposée combine des chemins générés dans les itérations précédentes pour produire des nouveaux chemins améliorants. Ces derniers ont généralement des degrés d'adjacence plus élevés que le degré associé à l'itération courante. L'algorithme fait preuve d'une haute performance contre la DP standard et aussi contre le MDDPA pour le même jeu de données utilisé dans la première contribution.

Les résultats obtenus ont montré que MDDPA et PAB constituent des outils de résolution

très efficaces, parfaitement adaptées à la méthode de CG. Ainsi, il s'est avéré nécessaire d'évaluer les atouts de ces nouvelles méthodes dans un processus global de CG. Pour ce faire, nous avons relevé deux grandes questions :

1. L'efficacité locale des méthodes primales proposées resterait-elle la même dans toutes les itérations de CG ?
2. Dans quelle mesure les effets secondaires, dus principalement à la qualité de la solution duale fournie par le MP, influenceraient-ils la performance de ces méthodes dans un cadre de CG ?

Afin de répondre à ces questions, nous proposons dans une troisième contribution d'intégrer les deux méthodes dans un algorithme de CG. Ainsi, nous introduisons d'abord un cadre général appelé *Primal Column Generation* (PCG) comme alternative au cadre standard de la CG. Le PCG représente la première implémentation des méthodes proposées dans un schéma général de CG. Dans cette implémentation, nous proposons également plusieurs adaptations de ces méthodes liées essentiellement au point initial et aux critères d'optimalité. D'un côté, le PCG dispense le solveur de tout besoin d'ajustement des problèmes à résoudre. D'un autre côté, contrairement aux méthodes heuristiques utilisées en pratique, le PCG se sert d'un nombre minimal de paramètres. Avec ces atouts, le solveur jouit d'une auto-capacité qui lui permet de décider d'arrêter la résolution des SPs chaque fois qu'il n'est pas nécessaire de continuer.

L'efficacité du PCG a été évaluée sur des instances de VCSP décrites par Haase *et al.* (2001). Nous résolvons la relaxation linéaire du problème en utilisant chacune des méthodes primales proposées pour le SP. Puis nous comparons les résultats avec ceux obtenus par un algorithme standard de CG qui fait appel à la DP communément utilisé pour résoudre les SP. Les résultats expérimentaux ont montré l'efficacité ainsi que la flexibilité de la PCG par rapport à un algorithme standard de CG. En particulier, les algorithmes proposés ont permis d'économiser une grande partie du temps investi pour résoudre les SPs. Cette performance a donné naissance à d'énormes économies au niveau du temps total de résolution.

CHAPITRE 5 ARTICLE 1 : A MULTIDIRECTIONAL DYNAMIC PROGRAMMING ALGORITHM FOR THE SHORTEST PATH PROBLEM WITH RESOURCE CONSTRAINTS

Le texte de ce chapitre est celui de l'article (Himmich *et al.*, 2018a) :

A Multidirectional Dynamic Programming Algorithm for the Shortest Path Problem with Resource Constraints

Ilyas Himmich, Issmail El Hallaoui, François Soumis
GERAD et École Polytechnique de Montréal
Département de Mathématiques et Génie Industriel
C.P. 6079, Succ. Centre-Ville
Montreal, Quebec, Canada H3C 3A7
ilyas.himmich, issmail.elhallaoui, françois.soumis@gerad.ca

publié dans les cahiers du GERAD et soumis à

European Journal of Operational Research

date de soumission : Janvier 2018

Abstract

The shortest path problem with resource constraints finds a least cost path between two nodes in a network while respecting constraints on resource consumption. The problem is mainly used as a subproblem inside column generation for crew scheduling and vehicle routing problems. The standard approach for the subproblems is based on dynamic programming. This class of methods is generally effective in practice when there are only a few resources, but it seems to be time-consuming for huge instances with many resources. To handle this problem, we propose a new exact primal algorithm called the multidirectional dynamic programming algorithm (MDDPA). The proposed approach splits the state space into small disjoint subspaces. These subspaces are sequentially explored in several iterations, where each iteration builds on the previous ones, to reduce the dimension of the subspaces to explore and to quickly generate better paths. Computational experiments on vehicle and crew scheduling instances show the excellent performance of our approach compared to the standard dynamic programming method. In particular, MDDPA is able to generate feasible paths with up to 90% of the optimal cost in less than 10% of the time required by standard dynamic programming. This feature is useful in column generation and may greatly reduce the computational effort, because we can stop the MDDPA solution process once columns with sufficiently negative reduced costs are obtained.

Keywords : Transportation, Shortest path problem with resource constraints, Column generation, Directions, Dynamic programming.

5.1 Introduction

The shortest path problem with resource constraints (SPPRC) aims to find a path between two nodes in a network (a source s and a destination d) at minimum cost while respecting restrictions called resource constraints. This problem was introduced by Desrochers & Soumis (1988a) as an extension of the classical unconstrained shortest path problem, and it has since attracted the attention of researchers from various domains.

5.1.1 Literature review

Several real-world applications have been modeled as SPPRCs ; e.g., military aircraft management (Zabarankin *et al.*, 2001), railroad management (Halpern & Priess, 1974), and service routing in communication networks (Xue, 2000). The problem also appears as a subproblem

in column generation (CG) for a huge variety of vehicle routing problems ranging from distribution problems to fleet assignment and crew scheduling such as bus driver scheduling (Desrochers, 1986) and airline crew pairing (Desaulniers *et al.*, 1997). CG is a well-known approach for large vehicle and crew scheduling problems (VCSPs). It is based on a decomposition of the generic formulation of the problem into a master problem (MP), which is generally a set partitioning problem with side constraints, and one or more subproblems. The latter are usually modeled as SPPRCs. They are used to feed potential columns to the MP at each CG iteration until an optimality criterion is satisfied. These columns are derived from feasible shortest paths with negative reduced costs. They may represent routes, schedules, or planning strategies, depending on the application.

Research into the SPPRC has explored different versions of the problem, with differing numbers of resources and applications. The three versions are : 1. The constrained shortest path problem (CSPP), where a single resource constraint is imposed at the destination node as an upper (or lower) bound on the cumulative consumption of the resource along the chosen path. 2. The shortest path problem with time windows (SPPTW), which also has a single resource but the restrictions are modeled at each node of the network : upper and lower bounds must be respected by every partial path. 3. SPPRC, which is a generalization of SPPTW that considers several resources. For the sake of generality, we consider the SPPRC.

Several solution methods have been proposed. These methods were grouped in Pugliese & Guerriero (2013b) into three main classes : k-path ranking methods, Lagrangean relaxation (LR) approaches, and dynamic programming (DP). The k-path ranking methods relax the resource constraints and compute, in increasing order of cost, the shortest paths between two nodes until a feasible path is identified. This approach was introduced in Handler & Zang (1980) and improved in Santos *et al.* (2007).

LR approaches relax the complex resource constraints before solving a Lagrangean dual problem. The efficiency of these methods depends on the strategies used to reduce the duality gap. Such methods have been developed by Handler & Zang (1980), Beasley & Christofides (1989), Melhorn & Ziegelmann (2000), and Carlyle *et al.* (2008).

The standard approach for SPPRC is based on DP. The basic algorithm was devised by Desrochers & Soumis (1988a) as an extension of the Ford–Bellman algorithm with the addition of resource constraints (Nagih & Soumis, 2006). The algorithm assigns states to each node, where each state at node i refers to a feasible partial path from s to i. It then repeatedly extends the states to generate new ones, and it stops when no further extension is possible. Moreover, dominance rules are applied to remove states corresponding to unpromising partial paths. DP-based algorithms for the SPPRC include Desrochers & Soumis (1988a,b); Desro-

siers *et al.* (1995); Desaulniers & Villeneuve (2000). These methods are particularly suitable for CG for three reasons. First, unlike LR, DP is able to provide the MP with several columns at a time. Second, it is able to deal with the most complex rules, usually modeled by nonlinear and even nonconvex resource constraints. Third, all the Pareto-optimal paths computed are integer by default. However, the application of DP to large transportation problems has shown that the method suffers from the curse of dimensionality. Many applications, especially in vehicle routing and crew scheduling, involve huge networks with hundreds of thousands of arcs and many resource constraints. DP algorithms create billions of labels during the solution process, and only a few of them are effective. This makes the process extremely time-consuming.

Intense research activity has explored the efficient solution of large problems. A modified version of the algorithm of Desrosiers *et al.* (1995), using preprocessing techniques and Lagrange multipliers, has been proposed by Dumitrescu & Boland (2003) for the CSPP. This problem has been addressed by Lozano & Medaglia (2013) using a pulse algorithm, this technique enumerates all possible paths and uses pruning strategies to narrow the search space. Nagih & Soumis (2006) have investigated the effect of the number of resources on the dominance rules ; they compute aggregated resources by projecting the original ones onto a space of smaller dimension. New refinements of the solution of the subproblems using DP have been presented by Feillet *et al.* (2007). They use a limited-discrepancy search to reduce the size of the state space. They use local rather than global search, and their algorithm extends the labels to a subset of the most promising arcs first, before allowing extension to less promising ones. Another improvement of the DP algorithm, called bidirectional dynamic programming (Righini & Salani, 2006), propagates labels both forward from the source to the destination and backward from the destination to the source. The forward and backward labels terminating at the same node are then combined to form a complete path.

5.1.2 Motivation and contributions

The motivation for this work is the observation that in CG, the purpose of solving subproblems is to feed the MP with new columns with sufficiently negative reduced costs ; they are not necessarily optimal, except in the final iterations. Therefore, we seek a primal method that returns sets of feasible solutions at different stages during the solution process of the subproblems. This feature is not offered by DP methods unless a heuristic stopping criterion is used, because they explore the entire state space before returning feasible solutions.

The contribution of this paper is threefold. First, we introduce a new formulation of the SPPRC that is suitable for reoptimization using previously generated labels. Second, we pro-

pose a new exact algorithm for the SPPRC, called the *multidirectional dynamic programming algorithm* (MDDPA). It is a primal method that returns sets of feasible paths of nonincreasing cost leading to optimality, while performing iterative searches on a reduced subnetwork. Third, we evaluate the performance of our approach compared to standard DP, the most common method in commercial solvers. The tests are performed on VCSP networks with up to 600,000 nodes and 1,000,000 arcs.

MDDPA combines three ideas :
i) A label storing procedure partitions the state space into small disjoint subspaces.
ii) Label loading strategies iteratively explore the subnetworks inducing these subspaces. We evaluate two loading strategies, nearest first and best first.
iii) Learning from locally efficient labels shows how to efficiently use the results of the previous MDDPA iterations. We use *label fathoming* to prevent the extension of unpromising states and *feasible descent directions* (FDDs) to construct new paths using those previously generated. The result is an efficient exact method that can quickly return interesting solutions and is able to prove optimality much earlier than standard DP can.

The paper is organized as follows. The next section presents the new generalized mathematical formulation of the SPPRC. Section 5.3 gives a detailed description of MDDPA. Section 5.4 reports the results of the computational experiments on simultaneous VCSP instances, and Section 5.5 provides concluding remarks.

5.2 Generalized mathematical formulation

Consider an acyclic connected network $G(V, A)$ where V is the set of nodes including the source and destination nodes s and d, and A is the set of arcs. Let $\mathcal{R}$ be the set of resources. For each arc $(i, j) \in A$, in addition to its cost c_{ij}, there is an $|\mathcal{R}|$-dimensional resource consumption vector $(r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|\mathcal{R}|})$. We denote by R_i^t the consumption of each resource $t \in \mathcal{R}$ over all the arcs composing a partial path π_i from s to i, while a_i^t and b_i^t are the lower and upper bounds on the resource $t \in \mathcal{R}$ at node i. The SPPRC finds a least cost path among all the paths from s to d that satisfy the resource constraints induced by $\mathcal{R}$.

5.2.1 Standard formulation

Let x_{ij} be the arc flow variable that takes the value 1 if the arc $(i,j) \in A$ is chosen to be a part of an optimal solution, 0 otherwise. The SPPRC can be formulated as follows :

$$(P_1) \quad \textit{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.1}$$

s.t.

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} -1 & \text{for } i = s \\ 0 & \forall\, i \in V \setminus \{s,d\} \\ 1 & \text{for } i = d \end{cases} \tag{5.2}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \quad \forall\, t \in \mathcal{R}, \forall\, (i,j) \in A \tag{5.3}$$

$$a_i^t \leq R_i^t \leq b_i^t \qquad \forall\, t \in \mathcal{R}, \forall\, i \in V \tag{5.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{5.5}$$

Constraints (5.2) are the flow conservation constraints. Constraints (5.3) model the resource consumption along arc (i,j) whenever it is part of the solution (path), while constraints (5.4) require the resource consumption along an *s-i* partial path to be within the corresponding resource interval. Note that it is allowed to arrive at a node $i \in V$ even if $R_i^t < a_i^t$ for some $t \in \mathcal{R}$; in this case R_i^t takes the value a_i^t. Constraints (5.5) are the binary requirements on the arc flow variables x_{ij}, $(i,j) \in A$.

In what follows, we reformulate the flow conservation constraints as set partitioning constraints. The purpose is twofold : first, the reformulation offers a new measure of distance between each node in the network and the destination node, and shows the length of each arc using this measure ; second, it allows us to introduce a new generalized formulation of the SPPRC. This formulation makes possible the partition of the state space that we propose, while the measure of distance is useful for exploring the elements of this partition.

5.2.2 Mathematical reformulation of the flow conservation constraints

We now reformulate constraints (5.2). We sort the nodes of $G(V,A)$ in topological order and refer to each node by its rank in this order. Thus, $V = \{1,2,...,|V|\}$ is the set of ordered nodes, where the source and destination nodes are indexed respectively by 1 and $|V|$. We define the notion of a *cocycle* as follows.

Definition 2. *Consider a node $k \in V \setminus \{|V|\}$. Let the k^{th} cocycle, denoted Co_k, be the set of all arcs $(i,j) \in A$ such that the origin i is topologically ordered before node k, and the destination j is ordered strictly after node k. Formally, $Co_k = \{(i,j) \in A | i \leq k < j\}$.*

With each cocycle, we associate the following cut called a cocycle constraint :

$$\sum_{(i,j)\in Co_k} x_{ij} = 1 \qquad \forall k \in \{1, 2, ..., |V| - 1\}. \tag{5.6}$$

Figure 5.1 shows the cocycle constraints on a four-node acyclic network. The next proposition shows that the cocycle constraints are sufficient to ensure connectivity of the path, i.e., flow conservation (of one unit) on the path.

Figure 5.1 Cocycle constraints

Proposition 1. *The cocycle constraints are equivalent to the flow conservation constraints in acyclic connected networks.*

Proof. Let $A^+(i)$ and $A^-(i)$ be the sets of arcs leaving and entering node i, respectively. For simplicity, we use x_a instead of x_{ij} to refer to the variable for an arc $a = (i,j)$. We have $Co_k = \cup_{i\leq k} A^+(i) \setminus \cup_{i\leq k} A^-(i), \forall k \in \{1, 2, ..., |V| - 1\}$. Thus, $\sum_{a\in Co_k} x_a = \sum_{i\leq k} \sum_{a\in A^+(i)} x_a - \sum_{i\leq k} \sum_{a\in A^-(i)} x_a, \forall k \in \{1, 2, ..., |V|\}$. For the nodes indexed by 1 and |V|-1, we have $Co_1 = A^+(1)$ and $Co_{|V|-1} = A^-(|V|)$. Thus, $\sum_{a\in Co_1} x_a = 1 \Longleftrightarrow \sum_{a\in A^+(1)} x_a = 1$, and $\sum_{a\in Co_{|V|-1}} x_a = 1 \Longleftrightarrow \sum_{a\in A^-(|V|)} x_a = 1$.

Let us prove the equivalence for any node $k \in \{2, ..., |V| - 1\}$.

- $\Rightarrow$ Suppose that $\boldsymbol{x}$ is a solution that satisfies the cocycle constraints but not the flow conservation constraints. Then $\exists\ p \in \{2, ..., |V| - 1\}$ such that $\sum_{a\in A^+(p)} x_a - \sum_{a\in A^-(p)} x_a = Q \neq 0$. We have :
  $\sum_{i\leq p}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a) = \sum_{i\leq p-1}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a) + \sum_{a\in A^+(p)} x_a -$

$\sum_{a\in A^-(p)} x_a$. Since $\sum_{a\in Co_k} x_a = \sum_{i\leq k}\sum_{a\in A^+(i)} x_a - \sum_{i\leq k}\sum_{a\in A^-(i)} x_a = 1, \forall k \in \{1,2,...,|V|-1\}$, we must have $1 = 1 + Q$, which is false unless $Q = 0$.

$\Leftarrow$ If the flow conservation constraints are satisfied by a solution $\boldsymbol{x}$, we have :
$\sum_{a\in Co_k} x_a = \sum_{i\leq k}\sum_{a\in A^+(i)} x_a - \sum_{i\leq k}\sum_{a\in A^-(i)} x_a$
$= \sum_{i\leq k}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a)$
$= \sum_{a\in A^+(1)} x_a - \sum_{a\in A^-(1)} x_a + \sum_{1<i\leq k}(\sum_{a\in A^+(i)} x_a - \sum_{a\in A^-(i)} x_a)$
$= 1$.
The cocycle constraints are then satisfied. Thus, the two formulations are equivalent.

□

Hence, the shortest path problem may be seen as a set partitioning problem with side constraints. Each column represents an arc, while the cocycle equality constraints ensure that every path covers each cocycle exactly once. Model (P_1) becomes :

$$
\begin{array}{lll}
(P_2) \quad \textit{Minimize} & \sum_{(i,j)\in A} c_{ij}x_{ij} & \\
\textit{s.t.} & & \\
& \sum_{(i,j)\in Co_k} x_{ij} = 1 \quad \forall k \in \{1,2,...,|V|-1\} & \\
& & \\
& (5.3)\text{–}(5.5) &
\end{array}
$$

5.2.3 Generalized mathematical formulation

The standard approach to the SPPRC is based on DP, as mentioned above. A DP algorithm associates with each partial path π_{si} from s to node i a state indicating the cumulative cost and resource consumptions. Each state is represented by an $(|\mathcal{R}|+1)$-vector $l = [C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|}]$ called a label. New labels are dynamically created by extending the existing ones. The labels are updated after each extension according to a resource extension function, $f_{ij} : \mathbb{R}^{|\mathcal{R}|+1} \to \mathbb{R}^{|\mathcal{R}|+1}$. The classical extension function is defined as follows : $f_{ij}(l) = l + [c_{ij}, r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|\mathcal{R}|}] = l'$.

An extension is valid only if the new label l' is feasible in terms of the resource constraints ; otherwise, it is eliminated. Feasible labels may be suppressed by the dominance rules. Dominance rules are used to compare each pair of feasible partial paths arriving at a given node ; the unpromising one is discarded.

Definition 3. *Let l_1 and l_2 be two feasible labels associated with two partial paths from s to node i. We say that l_2 is dominated by l_1 if and only if $C_1 \leq C_2$ and $R_1^t \leq R_2^t$ $\forall t \in \mathcal{R}$ and at least one inequality is strict.*

Definition 4. *A label l at node i is efficient if it is feasible and not dominated by any other label at node i at the end of the algorithm. A partial path is said to be efficient if it corresponds to an efficient label.*

Starting with an initial label $l_1 = [0, 0, ..., 0]$ at the source node s, and empty sets of labels at all other nodes, DP algorithms seek efficient labels by extending the partial paths for the existing efficient labels at a given node toward the outgoing arcs. The algorithm stops when all the efficient labels reach the destination node.

In what follows, we present a new generalized formulation of the SPPRC that will be useful for the understanding the working process of our MDDPA. This formulation assumes the existence of sets of feasible labels $\mathcal{S}_i$, $i \in V$ associated to feasible partial paths from the source node s to the nodes in the network. The way we construct these sets is described in section 5.3.2.1.

We denote by i_0 the index of the first node i in the topological order such that $\mathcal{S}_i \neq \emptyset$. The resulting model is a generalization of the classical SPPRC model, which considers only one label $l_1 = [0, 0, ..., 0]$ at the source node, and no labels at the remaining nodes, i.e., $\mathcal{S}_1 = \{l_1\}$ and $\mathcal{S}_i = \emptyset$, $\forall i \neq 1$. The model is as follows :

$$(P_3) \quad \textit{Minimize} \qquad \sum_{(i,j)\in A, i\geq i_0} c_{ij}x_{ij} + \sum_{i\in V, i\geq i_0, l\in\mathcal{S}_i} c_i^l y_i^l \tag{5.7}$$

s.t.

$$\sum_{i\geq i_0, l\in\mathcal{S}_i} y_i^l = 1 \tag{5.8}$$

$$\sum_{(i,j)\in Co_k} x_{ij} + \sum_{i>k, l\in\mathcal{S}_i} y_i^l = 1 \qquad \forall k \in \{i_0, i_0+1, ..., |V|-1\} \tag{5.9}$$

$$y_i^l(r_l^t - R_i^t) \leq 0 \qquad \forall i \in V, i \geq i_0, \forall l \in \mathcal{S}_i, \forall t \in \mathcal{R} \tag{5.10}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \qquad \forall t \in \mathcal{R}, \forall (i,j) \in A, i \geq i_0 \tag{5.11}$$

$$a_i^k \leq R_i^t \leq b_i^t \qquad \forall t \in \mathcal{R}, \forall i \in V, i \geq i_0 \tag{5.12}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, i \geq i_0 \tag{5.13}$$

$$y_i^l \in \{0,1\} \qquad \forall i \in V, i \geq i_0, \forall l \in \mathcal{S}_i \tag{5.14}$$

Here y_i^l is the decision variable that indicates whether or not the label l terminating at node i is used to construct an optimal path, c_i^l is its cumulative cost, and r_l^t is its cumulative consumption of resource t. We recall that each label in $\mathcal{S}_i$ represents a feasible partial path

from s to i that obviously covers all the cocycles Co_k for $k \in \{1, 2, ..., (i-1)\}$.

Constraint (5.8) ensures that exactly one label is selected from $\cup_{i \in V} \mathcal{S}_i$, so that its corresponding partial path is part of an optimal solution. In particular, if $i_0 = s$, the initial label $l_1 = [0, 0, ..., 0] \in \mathcal{S}_1$ may be the selected label. Constraints (5.9) ensure that all the cocycles $Co_k, k \geq i_0$ are covered exactly once. Each cocycle may be covered either by labels $l \in S_i, i \geq i_0$ or using an arc $(i, j) \in Co_k$. The cocycles Co_k, $k < i_0$ are obviously covered by the labels $l \in S_i$, $i \geq i_0$. Constraints (5.10) associate with R_i^t the cumulative consumption of resource $t \in \mathcal{R}$, if a label $l \in \mathcal{S}_i$ is chosen to be part of an optimal solution. Constraints (5.11) and (5.12) are subsets of constraints (5.3) and (5.4) with the restriction $i \geq i_0$. Constraints (5.13) and (5.14) are integrality constraints.

We observe that this formulation may be used to model different real-world situations. For example, re-optimization is often necessary when the network is affected by minor changes such as updates to the cost or resource consumptions of a subset of arcs, or the removal or addition of arcs or nodes. In particular, in the CG context, a subset of arcs will have reduced cost changes because of modifications to the dual values. In these cases, we have in hand a set of labels generated at previous CG iterations or during a previous solution process. Some of these labels are not affected by the minor perturbations and can be reused with little computational effort to produce new solutions. The affected part of the labels may be used in a heuristic framework after their attributes have been updated if possible.

Given that the labels in $\mathcal{S}_i$ for $i \in V$ correspond to feasible partial paths from the source node to a given node i, a first observation is that a feasible path from s to d can be constructed using a completion of any previously existing label at any node in the network, not only the initial label $l_1 \in \mathcal{S}_1$. Also, the extension of each label can be done independently of the rest of the labels. The formulation (P_3) then offers a disjoint partition of the solution space. These two observations are fundamental to MDDPA.

5.3 Solution approach

5.3.1 Motivation

The MDDPA is a generalization of the monodirectional search that characterizes classical DP algorithms, and the bidirectional search that was recently proposed for the elementary SPPRC Righini & Salani (2006). In fact, instead of extending labels in one direction starting from one initial label in the source node, or in two directions from the source and the destination nodes, the MDDPA provides a framework to extend labels using DP search starting from several nodes, where each node defines a new direction.

MDDPA can handle the problem of dimensionality without increasing the complexity in the worst case. The idea is to split the state space into several subspaces and to solve them iteratively while taking profit from the disjunction property offered by the generalized mathematical formulation (Section 5.2.3).

5.3.2 MDDPA algorithm

The core of our MDDPA approach is an initialization step in which we provide each node in the network with a set of efficient labels $\mathcal{S}_i$ (which may be empty). These labels are then iteratively loaded and extended from their resident nodes $i \in V$ to the destination node d, following predetermined loading rules.

MDDPA also allows the use of results from its previous iterations to tighten the dimension of the search subspace in the current iteration and to construct new solutions using previous ones. In fact, some of the previously generated labels are used to fathom unpromising labels in the current iteration, while others define FDDs that are useful for constructing new feasible paths. Therefore, MDDPA is based on three ideas : a label storing procedure, label loading strategies, and the ability to learn from locally efficient labels.

We need the following notation : k is the iteration number, and for each node $i \in V$, $\mathcal{S}_i$ is the set of stored labels ($\mathcal{S} = \bigcup_i \mathcal{S}_i$), $\mathcal{L}_i$ is the set of active labels, i.e., labels to extend ($\mathcal{L} = \bigcup_i \mathcal{L}_i$), and $\mathcal{P}_i$ is the set of locally efficient labels (Definition 5). In addition, Π_i is the subset of $\mathcal{P}_i$ that contains all the locally efficient labels that have contributed to the construction of a feasible path in a previous iteration. We denote by $A^+(i)$ the set of outgoing arcs from node i in $G(V, A)$. The procedure $LSP(\bar{G}, \mathcal{S})$ is the label storing procedure defined in Section 5.3.2.1. $LLS(\mathcal{L}, \mathcal{S}, i_0, k)$ is the procedure that loads the selected labels to extend from $\mathcal{S}$ at a given iteration k and returns the index i_0 of the first node in the topological order with $\mathcal{L}_i \neq \emptyset$; it is defined in Section 5.3.2.2. $Dominance(\mathcal{L}_i)$ is the dominance function (Definition 3) ; it applies the dominance rules at a given node i to fathom the unpromising labels and retain the efficient ones from the set of labels $\mathcal{L}_i$. The function $Extension(\mathcal{L}_i, j)$ extends the labels in $\mathcal{L}_i$ and returns the newly created labels at node j after checking their feasibility in terms of the cost bound and resource constraints (Section 5.2.3). $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$ is a procedure that fathoms unpromising labels and identifies FDDs using locally efficient labels in $\mathcal{P}_i$; this is explained in Section 5.3.2.3.

Finally, *CostBounding*(C^{best}) is the dynamic cost-bounding procedure ; it tightens the upper bounds at the nodes using the cost of the best path from Π_d. A label at node i can be extended to successor nodes only if its cost is at most the upper bound at node i. These upper bounds, denoted $\bar{C}_i$, are dynamically updated whenever a feasible path π with a

better cost is identified. They are computed at each node $i \in V$ as follows : $\bar{C}_i = C^{best} - C_i^{sp}$, where $C^{best} = min\{C_\pi, \pi \in \Pi_d\}$ is the cost of the best feasible path, and C_i^{sp} is the cost of the shortest path from the current node i to the destination node. The shortest paths $C_i^{sp}, i \in V$ are easily computed in a preprocessing step, with a backward call of the Ford–Bellman algorithm from the destination node.

Algorithm 5.1 presents the MDDPA pseudocode. We note that the *CostBounding(C^{best})* procedure is called at two levels of the MDDPA : at the end of each iteration, and inside the procedure $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$ if the latter found a path with a better cost than the existing best one. In addition to its ability to discard unpromising labels, the dynamic cost bounding ensures the generation of a sequence of feasible paths of nonincreasing cost leading to optimality.

Algorithm 5.1: MultiDirectional Dynamic Programming Algorithm

```
//Initialization //
Define a subnetwork Ḡ(V̄, Ā) of G(V, A)
Compute the reverse shortest path from d to s
C^best ← ∞
for all i ∈ V \ {1} do
   L_i ← ∅ ; S_i ← ∅ ; P_i ← ∅
S ← LSP(Ḡ, S)
k ← 1 , i_0 ← |V| − 1
//Search procedure //
repeat
   LLS(L, S, i_0, k)
   for i = i_0 to d do
      Dominance(L_i)
      LLEL(L_i, P_i, C^best)
      for all (i, j) ∈ A^+(i) do
         L_j ← L_j ∪ Extension(L_i, j)
      P_i ← P_i ∪ L_i
   CostBounding(C^best)
   k ← k + 1
until S = ∅
```

5.3.2.1 Label storing procedure (LSP)

The main purpose of LSP is to allow the SPPRC to be formulated as (P_3). We need to feed the sets of stored labels $\mathcal{S}_i, i \in V$ with labels representing feasible partial paths from s to i. Let $\bar{G}(\bar{V}, \bar{A})$ be a subnetwork of $G(V, A)$ defined as follows : $\bar{V} \subset V$, $\bar{A} \subset A$, $s \in \bar{V}$, $d \notin \bar{V}$ and for each arc $(i, j) \in \bar{A}$, we have $i \in \bar{V}$ and $j \in \bar{V}$. A node $j \in V$ is said to be a neighbor

of $\bar{G}$ if there is an arc $(i,j) \in A \setminus \bar{A}$ such that $i \in \bar{V}$. Algorithm 5.2 is used to fill the sets of stored labels associated with neighbors of $\bar{G}$. This algorithm assumes also that the nodes in $\bar{V}$ are topologically sorted.

Algorithm 5.2: Label Storing Procedure LSP($\bar{G}, \mathcal{S}$)

Initialization. $\mathcal{S}_i \leftarrow \emptyset$, $\forall i \in V$; $\mathcal{L}_1 \leftarrow \{[0,0,...,0]\}$; $\mathcal{L}_i \leftarrow \emptyset$ $\forall i \in V \setminus \{1\}$
for all $i \in \bar{V}$ **do**
  $Dominance(\mathcal{L}_i)$
  for all $(i,j) \in A^+(i)$ **do**
    $\mathcal{T}_j \leftarrow Extension(\mathcal{L}_i, j)$
    if $(i,j) \in \bar{A}$ **then**
      $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \mathcal{T}_j$
    else
      $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \mathcal{T}_j$
return $\mathcal{S}_i$ $\forall i \in V$

Starting from the initial label $l_1 = [0,0,...,0] \in \mathcal{L}_1$ at the source node, Algorithm 5.2 is a modified version of standard DP. It calls the dominance function at a node $i \in \bar{V}$, extends the efficient labels to the adjacent nodes $j \in V$ using arcs $(i,j) \in A^+(i)$, and saves the newly created labels in a temporary list $\mathcal{T}_j$. If arc $(i,j) \in \bar{A}$, $\mathcal{T}_j$ is added to $\mathcal{L}_j$, to be extended later. Otherwise, node j is a neighbor of $\bar{G}$, so $\mathcal{T}_j$ is immediately stored in $\mathcal{S}_j$. By the end of Algorithm 5.2, all the neighbors have sets of labels $\mathcal{S}_i$. This structure allows the SPPRC to be formulated as a (P_3) problem.

Remark 1. *Let $\mathcal{S}_i$, $i \in V$ be the set of labels generated by Algorithm 5.2. The sets of paths generated by the extension of labels in $\mathcal{S}_i$ $\forall i \in V$ are disjoint, in the sense that we cannot obtain the same path by extending two different labels in $\cup_{i \in V} \mathcal{S}_i$.*

We observe that $\bar{G}(\bar{V}, \bar{A})$ can be chosen to be any connected subgraph of G that fulfills the conditions mentioned in its definition. In a CG context, we construct $\bar{G}$ using the paths corresponding to the nondegenerate basic columns (variables) of the MP. In particular, we include in $\bar{G}$ all the nodes and arcs composing these paths except the destination node and the arcs entering to it. First, these paths cover in a balanced way all the regions of the network, since they form a feasible solution to the MP. Second, they have zero reduced costs, so we consider that their sequences of arcs and nodes are better able to contribute to the generation of new paths with negative reduced costs.

Proposition 2. *Using the sets of labels constructed by Algorithm 5.2, the formulation (P_1) is equivalent to (P_3), in the sense that every feasible solution for one is also feasible for the other. In particular, the two formulations have the same optimal solution.*

Proof. Let $\boldsymbol{x^\pi}$ be a feasible solution to (P_1), and π the corresponding path, there is a sequence of feasible labels in $G : \{l_i, i \in \mathcal{N}^\pi\}$, where $\mathcal{N}^\pi$ is the set of nodes traversed by π. Let $v \in \mathcal{N}^\pi$ be the first node of path π (in topological order) that is a neighbor of $\bar{G}$. The corresponding label $l_v = [c^{l_v}, r^1_{l_v}, r^2_{l_v}, ..., r^{|\mathcal{R}|}_{l_v}]$ is then feasible in G and therefore feasible in $\bar{G}$, so it is stored in $\mathcal{S}_v$ by Algorithm 5.2. If we set $y^{l_v}_v = 1$ and $y^l_i = 0, \forall l \neq l_v, \forall i \in V$, and we set R^t_v to $r^t_{l_v}$ for each $t \in \mathcal{R}$, constraints (5.10) and (5.12) are verified, and constraints (5.11) are verified for $k \leq v$. Moreover, (P_3) becomes a restricted problem of (P_1) with v as source node and l_v as initial label. So, $\boldsymbol{x^\pi}$ verifies obviously the rest of constraints.

Suppose now that $(\boldsymbol{x^\pi}, \boldsymbol{y^\pi})$ is a feasible solution to (P_3) and π the corresponding path. Let $\mathcal{A}^\pi$ be the set of arcs composing π. First, if we set $x_{ij} = 1\ \forall (i,j) \in \mathcal{A}^\pi$ and $x_{ij} = 0\ \forall (i,j) \notin \mathcal{A}^\pi$, constraints (5.2) are clearly verified. Second, there is necessarily a node $v \in V$ and a label $l_v \in \mathcal{S}_v$ such that $y^{l_v}_v = 1$ and $y^l_i = 0, \forall l \neq l_v, \forall i \in V$. Label l_v is feasible in $\bar{G}$, it is then feasible in G, so constraints (5.3) and (5.4) are verified for each $i \leq v$, and constraints (5.3) and (5.4) are verified for $i > v$ are the same as in (P_3). This completes the proof. □

On the one hand, Proposition 2 proves that solving (P_1) is equivalent to solving (P_3). On the other hand, Remark 1 shows that the sets of stored labels $(\mathcal{S}_i)_{i=1,...,|V|-1}$ offer a disjoint partition of the solution space of the SPPRC. These two results form the core of MDDPA, which iteratively solves (P_3), using the label loading strategies discussed below.

5.3.2.2 Label loading strategies (LLS)

With each set of labels $\mathcal{S}_i$ generated by Algorithm 5.2 at node i we associate a restricted search space induced by the subnetwork $G_i(V_i, A_i)$ of $G(V, A)$, where $V_i = \{j \in V, j \geq i\}$ and $A_i = \{(j,k) \in A, j \geq i\}$. These search spaces can be explored with various label loading strategies, and the order in which the sets of labels $\mathcal{S}_i, i \in V$ are extended is important since it may have a considerable impact on the effectiveness of the overall algorithm. We propose two loading strategies : *Nearest First* is based on the distance from the destination node, and *Best First* is based on the labels' costs.

5.3.2.2.1 Nearest first (NF) strategy

The NF strategy extends the labels $\mathcal{S}_i, i \in V$ while prioritizing labels that are closer to the destination node. The distance from the destination node is computed using the number of uncovered cocycles. Recall that labels in $\mathcal{S}_i$ correspond to partial paths that cover the cocycles $Co_1, Co_2, ..., Co_{i-1}$. When we add complementary sequences of arcs that cover the remaining cocycles $Co_i, Co_{i+1}, ..., Co_{|V|-1}$, we form complete paths from s to d.

Clearly, the labels stored at the nodes topologically ordered at the end of the network correspond to partial paths covering the largest portions of cocycles. They therefore need shorter sequences of arcs to form complete paths. Furthermore, extending these labels before those that are relatively far from the destination node could generate paths quickly, because the search spaces induced by the subnetworks $G_i(V_i, A_i)$ are of limited dimensions and so require limited computational effort.

The NF loading rule extends the labels in $\mathcal{S}_i$ node by node (or set by set) in reverse topological order of the nodes. Formally, the sets of labels in $\mathcal{S}_j$ are extended before those in $\mathcal{S}_i$ if $i < j$ (node i is topologically ordered before node j). Since there is no guarantee that the extension of $\mathcal{S}_i$ will lead to feasible paths, and since the extension of each set of labels individually may be inefficient, MDDPA extends several sets of labels at each iteration. Algorithm 5.3 gives the NF strategy.

Algorithm 5.3: $LLS(\mathcal{L}, \mathcal{S}, i_0, k)$ using NF strategy

$p_k \leftarrow p_0.r^k$
$i \leftarrow i_0 - 1$
repeat
  $\mathcal{L}_i \leftarrow \mathcal{S}_i$
  $\mathcal{S}_i \leftarrow \emptyset$
  $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_i$
  $i \leftarrow i - 1$
until $|\mathcal{L}| \geq p_k$
$i_0 \leftarrow i$

Algorithm 5.3 uses a sequence of jumps (p_k) to extend the selection of labels at each iteration k. The lengths of the jumps are defined using the geometric sequence $p_k = p_0.r^k$ with a first term p_0 and a common ratio $r > 1$. Formally, if i_0 is the first node in the topological order from which the labels were extended at iteration $k-1$, and p_k is the jump to perform at iteration k, the labels to load at the k^{th} iteration are $\mathcal{S}_{i_0-1}$, $\mathcal{S}_{i_0-2}$, $\mathcal{S}_{i_0-3}$,...,$\mathcal{S}_j$, such that $\sum_{i=j}^{i=i_0-1} |\mathcal{S}_i| \geq p_k$ and $\sum_{i=j+1}^{i=i_0-1} |\mathcal{S}_i| < p_k$. The use of this sequence makes the last jumps larger, which implies the exploration of large subspaces during the final iterations. This makes the algorithm more efficient for two reasons. First, as a primal method, MDDPA becomes rich in primal information during the final iterations. This information, extracted from the rapid early iterations, is used to tighten the cost bounds and to fathom large portions of unpromising labels, which can substantially reduce the dimensions of the subspaces in the final iterations and consequently reduce the computational complexity of these iterations. Second, we expect MDDPA to return an optimal solution before the end of the process. Therefore, it is always better to do large searches in the final iterations, since the purpose is

mainly to prove optimality.

5.3.2.2.2 Best first (BF) strategy

In the BF strategy, the labels to extend at each iteration are loaded according to their cumulative costs instead of their distance from the destination node. This allows the most promising labels, i.e., those with the lowest costs, to be extended first. These labels are expected to generate paths with interesting reduced costs. This is an appropriate strategy for CG, where columns with good reduced costs are sufficient, especially in the early iterations.

To determine which labels to extend at iteration k of MDDPA, we first sort the set of labels $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$ in increasing order of cumulative cost. Only the most promising labels are extended at each iteration. Furthermore, the lengths of the jumps to perform on $\mathcal{S}$ at iteration k are dynamically updated according to a geometric sequence $p_k = p_0.r^k$, where p_0 is the first jump and $r > 1$ is a common ratio. The stored labels that were extended from $\mathcal{S}$ in previous iterations are deleted so that they cannot be reused in the current iteration. Note that the jumps become longer between iterations, but the sizes of the subspaces to explore in the final iterations do not have the same rate of increase. This is because considerable portions of the labels are expected to be deleted by the label fathoming techniques (Section 5.3.2.3), which reduces the computational complexity. Algorithm 5.4 gives the BF loading strategy.

Algorithm 5.4: $LLS(\mathcal{L}, \mathcal{S}, i_0, k)$ using BF strategy

Sort $\mathcal{S}$ in increasing order of cost
$p_k \leftarrow p_0.r^k$
let $l^* = [C_{l^*}, R^1_{l^*}, R^2_{l^*}, ..., R^{|\mathcal{R}|}_{l^*}]$ be the label of rank p_k in $\mathcal{S}$
for all $i \in V$ **do**
  for all $l \in \mathcal{S}_i$ **do**
    if $C_l \leq C_{l^*}$ **then**
      $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{l\}$
      $\mathcal{S}_i \leftarrow \mathcal{S}_i \setminus \{l\}$
$i_0 = \text{argmin}_{\mathcal{L}_i \neq \emptyset} \mathcal{L}_i$

5.3.2.3 Learning from locally efficient labels (LLEL)

Learning refers to techniques that use available information to improve the solution process. This information can either be extracted from knowledge of the problem or generated at previous iterations. The efficiency of learning techniques depends on the quality of this

information and on the ability of the algorithm to improve this quality as the iterations proceed. MDDPA, a primal method, is able to dynamically update the primal information and improve its quality, whereas standard DP is not able to use such information.

In this section, we introduce techniques that use the previously generated labels to fathom unpromising labels and to construct feasible improving paths. We first give the following definition.

Definition 5. *A label l_i at node i is said to be locally efficient if it is efficient at a given iteration of MDDPA. The corresponding partial path is called a locally efficient partial path. The set of locally efficient labels at node i is denoted $\mathcal{P}_i$.*

5.3.2.3.1 Label fathoming using locally efficient labels

In practice, when we solve SPPRC using DP, the number of labels grows exponentially with the size of the network and the number of resources. Most labels correspond to partial paths that will not prove useful. We use the term label fathoming to refer to tools that recognize and stop the extension of unpromising labels.

Locally efficient labels are labels that have passed the dominance tests in a previous iteration and could potentially dominate new labels. The fathoming technique uses these labels to dominate new ones and therefore reduces the search space in the subsequent iterations. Proposition 3 justifies the validity of dominance using these labels and indicates their usefulness.

Proposition 3. *If label $l^{'} \in \mathcal{P}_i$ dominates $l \in \mathcal{L}_i$, then l can safely be discarded without affecting optimality.*

Proof. The proof is simple. If $l^{'}$ dominates l, then no extension of l will lead to a path better than that obtained while extending $l^{'}$ using the same extension function. If $l^{'}$ is dominated by another label $l^{''}$, then $l^{''}$ dominates l as well. Therefore, discarding l does not affect optimality. □

Proposition 4 shows that locally efficient labels are able to discard labels that were not eliminated by dynamic cost bounding.

Proposition 4. *Locally efficient labels are able to dominate new labels that were not eliminated by cost bounding.*

Proof. Suppose that C^{best} is the cost of the best feasible path found in a given iteration. Consider a label $l = [C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|}]$ at node i, and let C_i^{sfp} and C_i^{sp} be respectively the

cost of the shortest feasible partial path and the cost of the shortest unconstrained partial path from i to d. By definition, $C_i^{sfp} \geq C_i^{sp}$, so $C^{best} - C_i^{sfp} \leq C^{best} - C_i^{sp}$. If $C^{best} - C_i^{sfp} \leq C_l \leq C^{best} - C_i^{sp}$, then label l is unpromising and cannot be eliminated by cost bounding at node i. However, it can be dominated if there is a locally efficient label dominating it, as shown in Proposition 3. □

Recall that MDDPA also uses cost bounding to discard unpromising labels. Dominance using locally efficient labels has three main advantages. First, it involves the resource consumption of labels, while cost bounding is based only on the cost criterion. Second, the locally efficient labels have already resisted dominance and shown good potential during the previous iterations. They are efficient, and some of them have contributed to the construction of feasible paths. These effective sequences of arcs are therefore more reliable and better able to dominate other labels terminating at their resident nodes in subsequent iterations. Third, locally efficient labels are able to fathom unpromising labels regardless of the nature of the extension function, which may be nonlinear or even nonconvex, while linearity is essential for dynamic cost bounding. However, the strength of dynamic cost bounding is its speed, since it needs only one comparison test on the cost values while dominance with locally efficient labels requires the verification of $|\mathcal{R}|+1$ inequalities. When used together, the two fathoming techniques are able to identify and discard a large percentage of the labels that correspond to ineffective partial paths. The effectiveness of the two techniques depends on the quality of the available primal information.

5.3.2.3.2 Feasible descent directions

The motivation for this idea is the observation that at the end of each iteration we know the sequences of arcs that have contributed to the construction of feasible paths. These sequences are more likely to be part of new paths. FDDs aim to use this information efficiently to rapidly produce new paths. The costs of the previous paths are used to update the cost upper bounds at the nodes of the network. Moreover, the efficient labels that have produced these paths are also locally efficient labels, and they are therefore useful for dominating new labels. In this section we use these labels to define potential directions that may contribute to better paths.

Let $\pi \in \Pi_d$ be a feasible path generated at a given iteration of MDDPA. We use $\boldsymbol{x^\pi}$ to denote the corresponding $|A|-$solution vector. For each path $\pi \in \Pi_d$ traversing a set of nodes denoted $\mathcal{N}^\pi$, there is a set of locally efficient labels $\{l_i^\pi, i \in \mathcal{N}^\pi\}$ corresponding to the partial paths that have contributed to the construction of π. Let $\boldsymbol{x_i^\pi}$ be the $|A|-$vector

corresponding to the partial path l_i^π terminating at node $i \in \mathcal{N}^\pi$ such that $(\boldsymbol{x_i^\pi})_a = 1$ if arc a is part of the partial path associated with l_i^π, and $(\boldsymbol{x_i^\pi})_a = 0$ otherwise.

Definition 6. *Consider a feasible path π. We call direction the vector $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$ where $i \in \mathcal{N}^\pi$.*

The directions $\boldsymbol{d_i^\pi}$ store the arcs covering all the cocycles $Co_i, Co_{i+1}, ..., Co_{|V|-1}$. These arcs can be useful for completing the extension of new labels ending at node $i \in V$. Formally, consider a label l_i at node $i \in V$, and let $\boldsymbol{x_i^{l_i}}$ be its corresponding $|A|-$vector. Let $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$ be a direction induced by a previously generated path $\pi \in \Pi_d$ at node $i \in V$. The vector $\boldsymbol{x^{\pi'}} = \boldsymbol{x_i^{l_i}} + \boldsymbol{d_i^\pi}$ defines a complete path π' from s to d. There is no guarantee that the resulting path π' is feasible or that it improves the cost. A test of feasibility and cost improvement is needed, and we use previously available information to check whether or not these two properties are satisfied. Figure 5.2 illustrates the construction of new paths using directions.

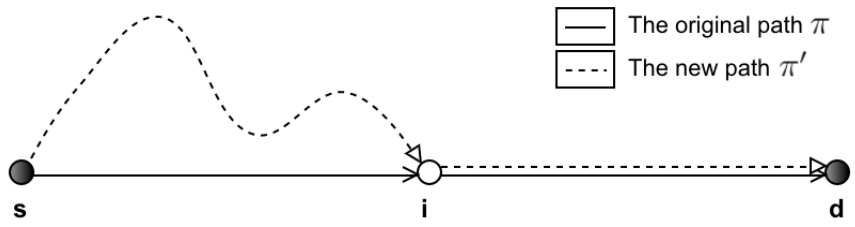

Figure 5.2 Notion of direction

Definition 7. *Consider a feasible path π of cost C_π and a direction $\boldsymbol{d_i^\pi}$ at node $i \in V$. Let $\mathcal{L}_i$ be a set of labels at node i. The direction $\boldsymbol{d_i^\pi}$ is said to be an FDD if there is a label $l_i \in \mathcal{L}_i$ such that the path π' given by $\boldsymbol{x^{\pi'}} = \boldsymbol{x_i^{l_i}} + \boldsymbol{d_i^\pi}$ is feasible and $C_{\pi'} \leq C_\pi$ where $C_{\pi'}$ is the cost of π'. If we also have $C_{\pi'} \leq C^{best}$, then $\boldsymbol{d_i^\pi}$ is called an improving descent direction (IDD).*

Let π be a feasible path and l_i^π the locally efficient label at node $i \in V$ whose extension gave π. Consider a set of labels $\mathcal{L}_i$ at node i. The following proposition gives a sufficient condition that makes a direction $\boldsymbol{d_i^\pi}$ an FDD.

Proposition 5. *Consider a label $l \in \mathcal{L}_i$ and a feasible path $\pi \in \Pi_d$. If l dominates l_i^π, then $\boldsymbol{d_i^\pi} = \boldsymbol{x^\pi} - \boldsymbol{x_i^\pi}$ is an FDD.*

Proof. Consider a label $l = [C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|}] \in \mathcal{L}_i$ and a locally efficient label $l_i^\pi = [C_{l_i^\pi}, R_{l_i^\pi}^1, R_{l_i^\pi}^2, ..., R_{l_i^\pi}^{|\mathcal{R}|}]$ corresponding to a feasible path π of cost C_π. If l dominates l_i^π, then $C_l \leq C_{l_i^\pi}$ and $R_l^t \leq R_{l_i^\pi}^t \; \forall t \in \{1, 2, ..., |\mathcal{R}|\}$. Since π is feasible, the path π' defined by

$\boldsymbol{x}^{\boldsymbol{\pi}'} = \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{l}} + \boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}$ is feasible. Moreover, $C_{\pi'} = C_l + cost(\boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}) \leq C_{l_i^\pi} + cost(\boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}) = C_\pi$. The resulting path π' is then a feasible path of better cost. This implies that the direction $\boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}$ is an FDD. □

In this case, the new path π' is added to the set of available paths Π_d to enrich the primal information. Otherwise, if the locally efficient label l_i^π dominates $l \in \mathcal{L}_i$, the latter is not able to provide a better feasible path and can be safely eliminated from $\mathcal{L}_i$, as shown in Proposition 3.

Remark 2. *Let $\boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}$ be an FDD and l_i^π the locally efficient label defining it. If the path with the best cost was found using the extension of l_i^π, then $\boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}$ is an IDD.*

If no label in $\mathcal{L}_i$ dominates l_i^π, MDDPA must seek a new FDD using the extension of the sets of active labels $\mathcal{L}_i, i \in V$ by calling a DP search in the restricted search space defined by the predetermined label loading strategy. When the dominance test fails, in the sense that no label dominates another, then if the cost of one of the labels $l \in \mathcal{L}_i$ is less than the cost of l_i^π, the path π' given by $\boldsymbol{x}^{\boldsymbol{\pi}'} = \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{l}} + \boldsymbol{d}_{\boldsymbol{i}}^{\boldsymbol{\pi}}$ is improving, but we still need to check whether or not it is feasible.

Algorithm 5.5 identifies the descent directions.

Algorithm 5.5: $LLEL(\mathcal{L}_i, \mathcal{P}_i, C^{best})$

```
for all l ∈ L_i do
  for all l' ∈ P_i do
    if l' dominates l (label fathoming) then
      L_i ← L_i \ {l}
    if l dominates l' then
      if l' ∈ Π_i (FDD) then
        Let π be the feasible path corresponding to l'
        l_i^π ← l'
        d_i^π = x^π − x_i^π (new FDD d_i^π defined)
        x^π' = x_i^l + d_i^π (new feasible path π' defined)
        Π_d ← Π_d ∪ {π'}
        if C_π' < C^best (IDD) then
          C^best ← C_π
          CostBounding(C^best)
```

Clearly, finding FDDs does not require significant additional computational effort. The process may quickly return paths that are able to enrich the primal information, tighten the cost upper bounds, and accordingly strengthen the label fathoming.

It is interesting to note that matching two partial paths to construct a complete path was first proposed by Righini & Salani (2006) in their bidirectional DP algorithm for the elementary SPPRC. This idea was used later by Feillet *et al.* (2007) in order to take profit from the paths related to the positive valued variables in the current solution of the MP. The MDDPA offers a different framework for the application of this idea : first, new locally efficient labels are generated at each iteration, so feasible paths are constructed dynamically using these labels ; second, the feasibility is guaranteed by a simple dominance test regardless the nature of the resource constraints ; third, locally efficient labels serve also to stop the extension of non promising labels.

Proposition 6 shows that MDDPA is an exact solution approach.

Proposition 6. *MDDPA terminates by finding an optimal solution to the SPPRC.*

Proof. Let π be an optimal path to the SPPRC traversing the set of nodes $\mathcal{N}^\pi$. There is a sequence of efficient labels $\{l_i^\pi, i \in \mathcal{N}^\pi\}$ corresponding to π. Using Proposition 2, there is necessarily a node $i \in V$ such that $l_i^\pi \in \mathcal{S}_i$. Suppose that l_i^π is extended at iteration k of MDDPA. The labels $\{l_j^\pi, j > i, j \in \mathcal{N}^\pi\}$ are all efficient in G, therefore they are locally efficient at iteration k. The fact that all stored labels in $\mathcal{S} = \bigcup_i \mathcal{S}_i$ are extended by MDDPA completes the proof. □

5.4 Experimentation

5.4.1 Test instances

Our test instances are derived from the well-known VCSP in urban mass transportation systems. The aim of the VCSP is to simultaneously construct bus and crew schedules that cover a set of bus trips at a minimum cost, while satisfying a set of constraints over a 1-day horizon. These constraints impose the regulations of the collective agreements. A VCSP instance is defined in an acyclic network with a set of bus trips, where each line is composed of a predefined number of tasks. A task is a segment of trips that must be covered exactly once by a bus and a driver. A bus schedule is a sequence of tasks and deadheads. A deadhead is a trip without passengers that repositions the bus. A crew schedule, also called a *driver's duty*, is a working day composed of a sequence of tasks, deadheads, and breaks. The construction of these schedules is time-consuming because a driver can leave a bus trip at locations called *relief points* between two consecutive tasks. The higher the number of relief points, the more difficult the problem.

The VCSP is an NP-hard problem that is solved by CG. The MP is mostly a set partitioning

problem, where each task is assigned to exactly one driver and one bus, and the number of buses does not exceed the number available. Similarly, each column is associated with a possible driver's duty. The subproblems (SPs) are mainly SPPRCs, they generate schedules with negative reduced costs that satisfy the resource constraints. For the VCSP, we have seven resource constraints and two types of driver's duties that differ in the number of allowed bus changes during the same duty. This gives rise to two SPs, a SP per duty type. Indeed, similarly to Haase *et al.* (2001), the computational study consists only on the generation of crew schedules, since the bus schedules can be derived afterwards in polynomial time.

In CG, the cost distribution on the arcs changes from iteration to iteration. The costs are reduced costs computed using the dual values of the MP constraints, and they depend on the columns present in the MP at the given iteration. For a fair comparison of MDDPA and DP we must use instances with the same reduced-cost structure. For this reason, we perform tests on SP instances extracted from some of the CG iterations.

The VCSP instances (Table 5.1) were randomly generated using the generator of Haase *et al.* (2001). These instances differ in the number of bus trips considered (120, 160, 200, 240) and the number of relief points (5,7,9) ; the latter is one less than the number of tasks per bus trip. There are 12 possible configurations defining 12 classes of instances, denoted *rp_bl* where *rp* is the number of relief points and *bl* is the number of bus trips. For each class we generated five VCSP instances by varying the seed. These 60 instances were run using a CG solver, and for each instance, we captured two SPs, the first from the first 20% of the iterations and the second from the final 20%. We did this to evaluate our method and DP both at the beginning and at the end of the CG. This gives a total of 120 SP instances. The instances taken from the beginning of CG, called "b-instances," are indicated by "_b" in the notation of the instance class. Those taken from the end of the CG, called "e-instances," are indicated by "_e."

Table 5.1 List of test instances (MDDPA)

Instance class	# Nodes	# Arcs	# Relief points	# Tasks
5_120	50690.0	78989.2	5	120
5_160	91458.4	141286.0	5	160
5_200	143107.2	220018.0	5	200
5_240	205377.4	314692.6	5	240
7_120	98891.2	152192.4	7	120
7_160	178498.7	273065.2	7	160
7_200	280649.0	427856.4	7	200
7_240	402557.6	612310.2	7	240
9_120	162862.0	249093.6	9	120
9_160	295783.2	450255.6	9	160
9_200	463421.0	703604.8	9	200
9_240	665201.0	1008197.8	9	240

The experiments were performed on a MacBook Pro with a 2.5 GHz processor (Intel Core i5) and 4 Gb of RAM. MDDPA was implemented in C++ using Boost Graph library, a well-known C++ library. We compare MDDPA with a standard DP (std. DP). This algorithm is a well-tailored labelling algorithm provided by C++ Boost Graph library. In addition, it was enhanced with the same improvements used for the MDDPA. First, we sort the nodes of the network in a topological order, so the exploration of nodes by std. DP follows this order. Second, we associate to each node a cost upper bound computed as a preprocessing step using *Cost_Bounding* procedure. Finally, we tighten the resource upper bounds for all the nodes of the network by computing the reverse shortest paths from the destination node to each node for each resource. The topological order improves the std. DP search in acyclic networks, while the cost and resource bounding allows the pruning of a huge number of labels. Indeed, using these improvements, the considered std. DP forms the state-of-the-art DP algorithm for solving the SPPRC.

5.4.2 MDDPA vs. DP

For the 120 SP instances, MDDPA has proven itself against DP using either the NF or BF loading strategy. Tables 5.2 and 5.3 give a comparison of std. DP, MDDPA using NF, and MDDPA using BF, for the b-instances and e-instances respectively. We report for each approach the total time consumed in seconds (CPU), the number of labels created (# Lab.), and the number of calls to the dominance function (# Dom. calls). For NF and BF we give the time when we find an optimal solution for the first time (Opt. time). The results reported for each class are aggregated values of the results for the five instances.

In terms of solution time, Tables 5.2 and 5.3 show that MDDPA outperforms std. DP at

Table 5.2 Results for b_instances

Instance class	std. DP			NF				BF			
	CPU	# Lab.	# Dom. calls	CPU	Opt. time	# Lab.	# Dom. calls	CPU	Opt. time	# Lab.	# Dom. calls
5_120_b	3.56	7.15E+05	1.32E+07	2.21	1.05	2.54E+05	3.43E+06	2.44	2.15	4.06E+05	5.37E+06
5_160_b	9.64	1.69E+06	4.23E+07	5.25	2.58	5.57E+05	1.11E+07	4.84	3.24	6.66E+05	1.10E+07
5_200_b	21.53	3.29E+06	1.09E+08	10.93	6.67	1.21E+06	4.17E+07	13.59	11.97	1.65E+06	5.82E+07
5_240_b	33.65	5.56E+06	1.86E+08	21.01	16.42	2.51E+06	7.78E+07	24.97	10.52	3.12E+06	1.05E+08
7_120_b	8.84	1.63E+06	3.80E+07	4.76	3.22	5.37E+05	8.08E+06	6.27	4.59	9.24E+05	1.41E+07
7_160_b	18.93	3.63E+06	1.00E+08	15.30	10.63	1.92E+06	5.76E+07	14.54	8.79	2.01E+06	5.45E+07
7_200_b	51.39	7.81E+06	3.07E+08	33.90	20.76	3.66E+06	1.62E+08	47.80	34.74	5.13E+06	2.67E+08
7_240_b	89.38	1.33E+07	5.62E+08	53.45	30.35	5.26E+06	2.24E+08	62.83	29.92	6.68E+06	2.99E+08
9_120_b	15.29	3.02E+06	7.65E+07	9.71	5.97	1.09E+06	2.20E+07	10.97	8.04	1.51E+06	2.50E+07
9_160_b	41.98	7.24E+06	2.35E+08	24.95	14.99	2.53E+06	6.70E+07	22.73	9.85	2.65E+06	6.98E+07
9_200_b	93.57	1.41E+07	5.94E+08	70.68	46.90	6.64E+06	2.80E+08	86.90	44.49	8.06E+06	4.72E+08
9_240_b	176.64	2.37E+07	1.09E+09	114.91	61.69	1.09E+07	5.41E+08	132.42	19.63	1.24E+07	7.28E+08

both the beginning and the end of the CG. Figure 5.3 presents the time reduction factors obtained as a ratio of std. DP time to NF and BF time. It shows that the factors achieved by the two loading strategies are significant, especially for the e-instances. The BF strategy has a reduction factor for the b-instances ranging between 1.08 and 1.99 with an average of 1.44. This ratio is greater for the e-instances : up to 6.81, with an average of 3.33. The NF strategy is better. For b-instances, the total time is reduced by a factor varying between 1.24 and 1.97, with an average of 1.62. For e-instances, it is up to 6.87, with an average of 3.59. These differences between the results of b-instance and e-instance are in fact due to the quality of the dual values used to compute the reduced costs of the arcs. Indeed, it is known that these dual values are more stabilized at the end of CG. Thus, MDDPA becomes more and more efficient with time in a CG context.

Table 5.3 Results for e_instances

Instance class	std. DP			NF				BF			
	CPU	# Lab.	# Dom. calls	CPU	Opt. time	# Lab.	# Dom. calls	CPU	Opt. time	# Lab.	# Dom. calls
5_120_e	1.18	2.81E+05	1.49E+06	0.17	0.17	2.37E+03	1.36E+04	0.19	0.19	2.37E+03	1.36E+04
5_160_e	3.37	7.64E+05	7.42E+06	1.38	0.49	1.25E+05	1.66E+06	1.55	0.50	1.39E+05	1.73E+06
5_200_e	6.97	1.53E+06	2.20E+07	4.05	1.26	4.62E+05	9.19E+06	5.13	2.10	6.27E+05	1.14E+07
5_240_e	12.17	2.43E+06	3.89E+07	3.91	2.34	2.79E+05	3.73E+06	4.14	2.19	2.98E+05	4.24E+06
7_120_e	2.32	5.43E+05	2.74E+06	0.34	0.34	3.15E+03	1.39E+04	0.34	0.34	3.15E+03	1.39E+04
7_160_e	8.77	1.84E+06	2.44E+07	2.36	0.95	2.28E+05	2.50E+06	2.55	0.87	2.61E+05	3.07E+06
7_200_e	18.17	3.75E+06	6.45E+07	6.14	3.45	6.07E+05	1.28E+07	7.88	4.46	9.77E+05	1.76E+07
7_240_e	26.89	5.18E+06	9.07E+07	8.98	6.53	6.36E+05	9.04E+06	7.80	5.20	5.79E+05	7.92E+06
9_120_e	5.69	1.26E+06	1.11E+07	1.48	0.69	1.04E+05	5.54E+05	2.33	1.73	2.46E+05	1.60E+06
9_160_e	16.19	3.29E+06	4.42E+07	7.10	4.13	6.64E+05	8.51E+06	7.77	4.70	9.19E+05	1.30E+07
9_200_e	42.38	7.67E+06	2.13E+08	12.83	4.41	1.20E+06	1.51E+07	13.00	5.17	1.32E+06	2.24E+07
9_240_e	56.54	1.07E+07	2.18E+08	18.20	17.76	1.31E+06	2.32E+07	16.43	13.34	1.19E+06	1.84E+07

The time reductions achieved by MDDPA are induced by huge reductions in the number of labels. For b-instances, NF and BF give an overall label reduction factor of 2.24 on average ;

for e-instances this factor is on average 30.25 for NF and 29.18 for BF. A direct consequence is a reduction in the number of dominance calls. For b-instances, the dominance reduction factor ranges between 1.15 and 3.84 for BF, while it is up to 4.71 with an average of 2.89 for NF. For e-instances this factor is on average 31.46 for BF and 33.18 for NF. These reductions are a result of the quality of the labels generated during the first iterations of MDDPA. This primal information is efficiently used by the dynamic cost bounding and dominance with locally efficient labels. Many labels are fathomed, greatly reducing the time spent on dominance tests.

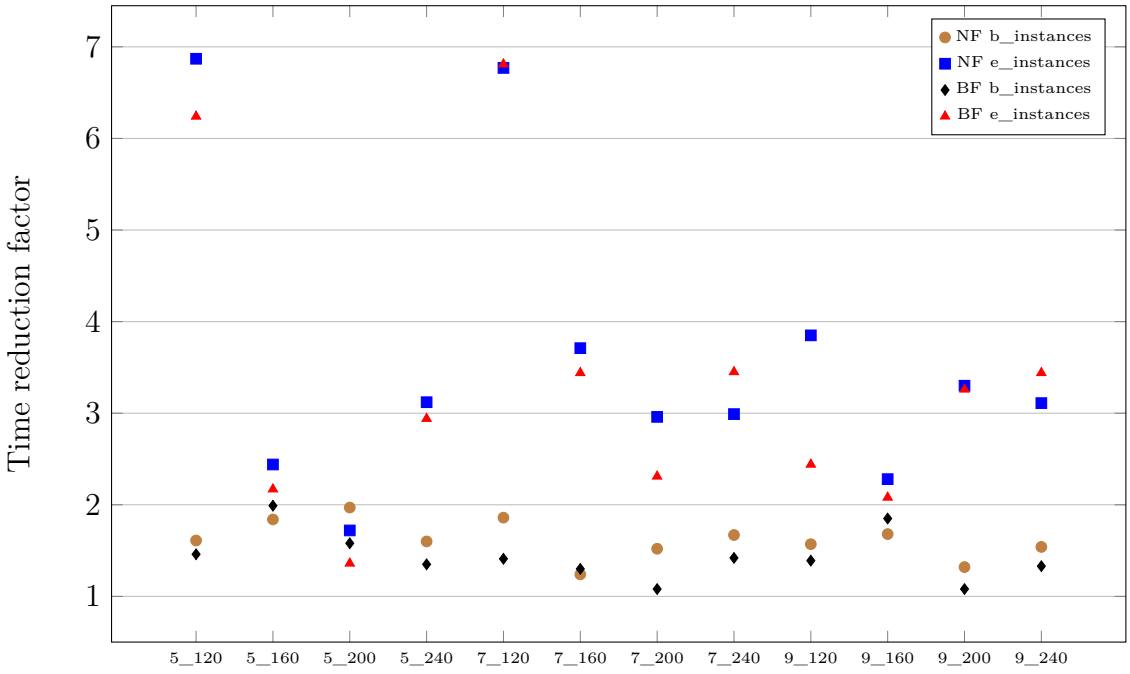

Figure 5.3 Time reduction factor (MDDPA vs. std. DP)

5.4.3 NF vs. BF

Tables 5.4 and 5.5 give a comparison of NF and BF, for the b-instances and e-instances respectively. We report, for each inctance class and each loading strategy, the averages of : the number of iterations (# Its.) ; the number of descent directions found during the solution process (#FDD) ; the percentage of the cost decrease attributed to these FDDs (% ACD by FDD) with respect to the cost of the previous paths from which the directions are constructed ; the percentage of the cost improvement attributed to IDDs (% Imp. of IDD) ; and the percentage of the std. DP time required for MDDPA to find an optimal solution (% Time in opt.). For the b-instances, we give the percentage of the optimal value returned by MDDPA

in less than 10% of the std. DP time (% Opt. in 10%). This sheds light on one of the most interesting feature of MDDPA : its ability to quickly find good solutions, especially in the first CG iterations.

Tables 5.4 and 5.5 indicate the excellent quality of solutions returned by NF and BF in a small proportion of the time consumed by DP. For the b-instances, NF returns an optimal solution in 26.77% to 56.18% of the std. DP time, with an average of 39.01%. BF requires 11.11% to 60.48% of the std. DP time, with approximately the same average. BF finds the optimal solution faster than NF does, especially for instances with more than 160 bus trips and 7 or 9 relief points. For the e-instances, the average time to find an optimal solution is 17.89% for NF and 20.23% for BF.

For the b-instances, in less than 10% of the std. DP time, NF finds paths with a cost ranging between 75.65% and 92.82% of the optimal value, and for BF the range is 45.11% to 69.56%. Figures 5.4 and 5.5 track the evolution of the objective value over time for NF and BF on the largest b-instance and e-instance respectively.

Table 5.4 MDDPA for b_instances

Instance class	NF						BF					
	# Its.	# FDD	% ACD by FDD	% Imp. of IDD	% Time in opt.	% Opt. in 10%	# Its.	# FDD	% ACD by FDD	% Imp. of IDD	% Time in opt.	% Opt. in 10%
5_120_b	8.0	17.2	26.0	15.7	29.6	82.2	3.0	27.8	20.1	7.9	60.5	46.9
5_160_b	8.0	62.0	21.6	13.0	26.8	85.4	3.6	356.0	16.3	4.8	33.6	59.3
5_200_b	7.8	255.3	15.1	23.2	31.0	88.4	4.3	2061.3	9.4	15.0	55.6	50.0
5_240_b	8.0	366.6	15.7	22.7	48.8	82.6	5.0	1830.0	10.9	14.6	31.3	50.8
7_120_b	7.6	71.0	24.1	10.1	36.4	75.6	4.0	261.0	19.5	5.5	51.9	45.1
7_160_b	8.0	362.5	11.5	3.3	56.2	81.6	4.0	324.2	13.1	2.9	46.4	66.6
7_200_b	7.8	282.3	15.2	17.6	54.0	86.0	5.0	2560.3	10.9	21.5	52.9	54.5
7_240_b	8.0	538.0	9.4	9.6	36.3	80.5	5.3	4556.5	8.1	8.1	25.3	59.5
9_120_b	7.6	149.4	23.8	13.8	39.1	81.9	4.0	79.6	18.0	10.5	52.6	56.4
9_160_b	8.0	247.4	23.7	13.3	35.7	79.0	4.4	1067.0	27.2	18.9	23.5	64.7
9_200_b	8.0	648.0	13.7	15.9	39.4	92.8	5.7	3843.0	7.7	16.2	33.8	56.5
9_240_b	8.0	698.8	13.4	18.3	34.9	85.0	6.0	4170.8	10.1	18.7	11.1	69.6

These results are useful in a CG context. The SPs aim to quickly provide the MP with feasible paths (columns) with good reduced costs that are not necessarily optimal. This is especially important in the first CG iterations, where the dual values are not yet stable, and the goal is to quickly feed the MP with more columns to stabilize the dual-value distribution. The need to prove optimality arises only during the final iterations.

Table 5.5 MDDPA for e_instances

Instance class	NF					BF				
	# Its.	# FDD	% ACD by FDD	% Imp. of IDD	% Time in opt.	# Its.	# FDD	% ACD by FDD	% Imp. of IDD	% Time in opt.
5_120_e	1.0	0.0	0.0	0.0	14.6	1.0	0.0	0.0	0.0	16.0
5_160_e	3.6	1.4	21.1	5.8	14.6	3.2	1.4	21.1	5.8	14.8
5_200_e	4.0	36.3	34.7	13.9	18.1	4.0	41.3	33.1	21.2	30.2
5_240_e	3.4	38.8	30.2	8.6	19.2	3.2	29.2	33.0	5.7	18.0
7_120_e	1.0	0.0	0.0	0.0	14.8	1.0	0.0	0.0	0.0	14.7
7_160_e	2.6	57.0	13.7	0.0	10.8	2.6	56.6	14.7	0.8	9.9
7_200_e	3.8	108.2	34.8	10.7	19.0	3.6	121.0	35.9	10.7	24.6
7_240_e	3.4	44.0	19.5	14.0	24.3	3.2	44.8	19.2	9.3	19.3
9_120_e	2.6	9.4	49.4	35.3	12.1	2.4	4.6	48.8	35.3	30.5
9_160_e	3.4	40.8	18.5	11.9	25.5	3.2	8.2	22.5	11.9	29.0
9_200_e	4.0	156.6	35.3	41.6	10.4	3.4	144.4	28.7	20.5	12.2
9_240_e	4.0	180.0	35.5	6.5	31.4	4.0	203.5	47.4	19.0	23.6

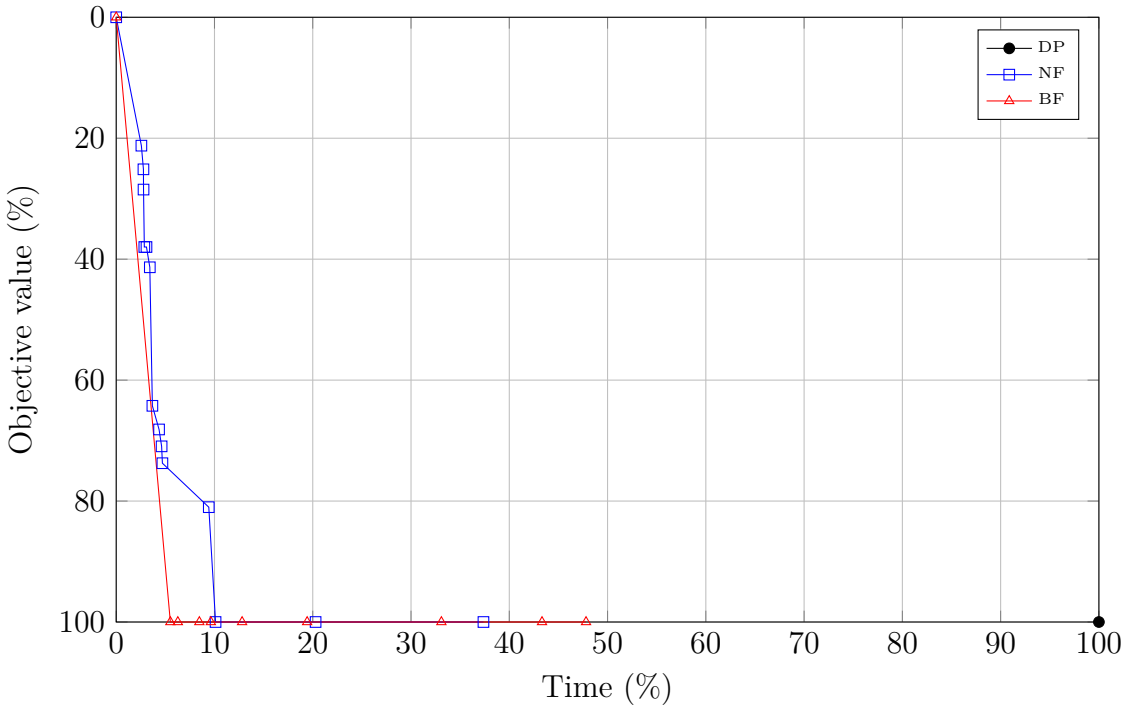

Figure 5.4 Improvement in objective value as function of time for b_instances

We wish to highlight the impact of the FDDs and IDDs on NF and BF ; these directions are used to construct better feasible paths. FDDs and IDDs are found for all the b-instances. The number of paths generated by FDDs is on average 291.14 for NF and 1663.25 for BF. For e-instances, the number of FDDs is 56 on average for NF and BF. FDDs are thus more effective in the b-instances, especially with BF. This may be because at the end of the CG process, the variation in the reduced costs on the arcs becomes small. Consequently, it is harder to find directions that are able to improve the previously generated paths. Moreover, in e-instances the SPs are tighter, so the number of paths with negative reduced costs is smaller, and consequently the probability of finding such paths decreases.

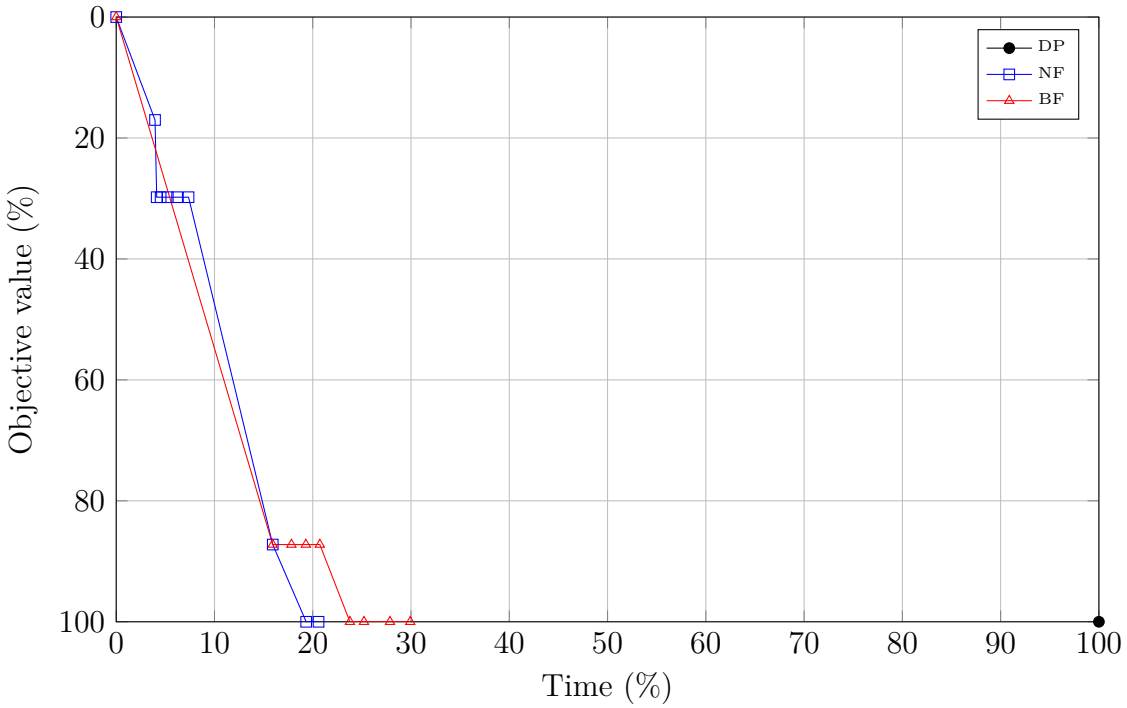

Figure 5.5 Improvement in objective value as function of time for e_instances

IDDs improve the cost by 14.65% on average for b-instances and 12.35% for e-instances. These values are slightly lower for BF. Figure 5.6 shows the impact of these directions on MDDPA, for one large instance from class 9_240. Here an optimal solution is found in less than 10% of the total time when IDDs are used, and in 25% of the total time when IDDs are not used. We conclude that IDDs are able to considerably decrease the objective value with just a little additional computational effort, and to tighten the cost upper bounds, reducing the number of labels and improving the total time.

MDDPA outperforms the std. DP for all the instances. It reduces the overall solution time

by a factor of up to 3.5. Furthermore, as a primal method, it can generate interesting feasible paths in a limited solution time. It is expected to lead to huge improvements in the overall CG solution time. NF can quickly make small improvements to the objective value, and it proves optimality earlier than BF does. However, BF converges more quickly to an optimal solution, especially for the most difficult b-instances. To summarize, either strategy can be used in the first CG iterations depending on the quality of columns deemed sufficient by the modeler, and NF is more appropriate for the final iterations, when we must prove optimality.

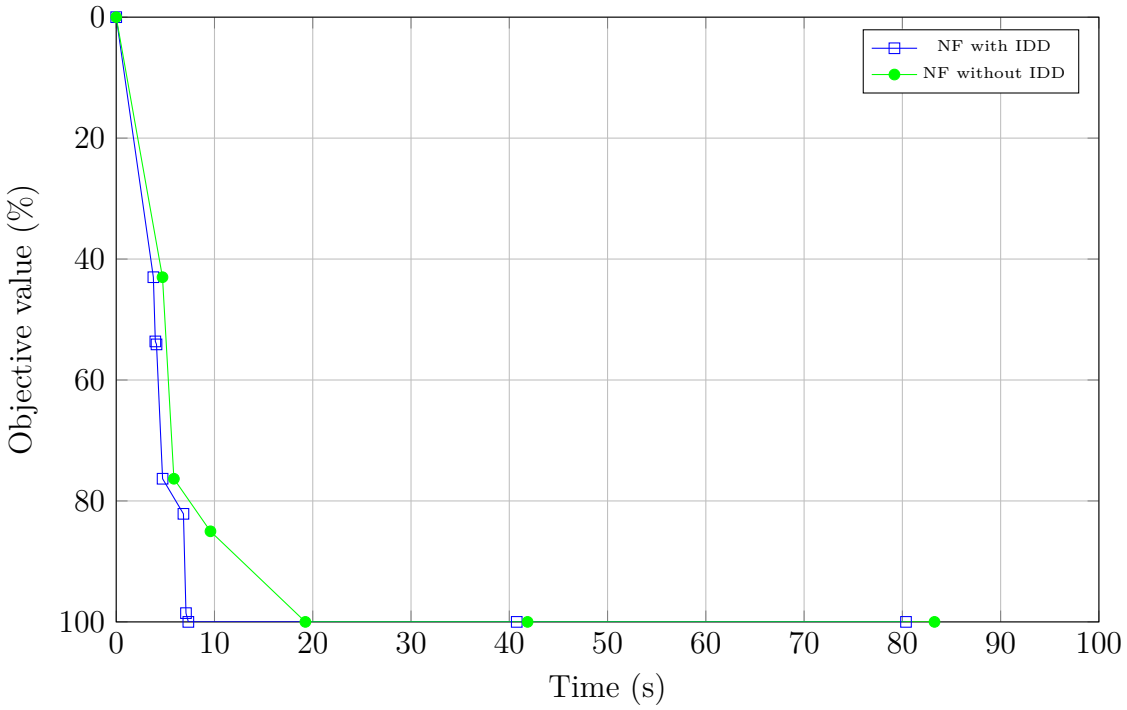

Figure 5.6 Effect of IDDs on MDDPA

5.5 Conclusion

We have presented a new solution approach for the SPPRC. MDDPA combines three techniques that work together to deal with the problem of dimensionality and to overcome the weaknesses of DP algorithms. The algorithm performs a disjoint partition of the search space using a label storing procedure. It then uses two loading strategies for the iterative exploration of a sequence of restricted search spaces. These two techniques enable the construction of a primal framework that allows the current iteration to take advantage of previous ones. In particular, our label fathoming techniques discard a huge number of labels. Also, the LLEL

procedure allows the detection of directions that help to provide the solver with good new paths.

We evaluated the algorithm on instances of the simultaneous VCSP, extracted from different stages of the CG process. MDDPA performs considerably better than std. DP, especially in the final CG iterations. Furthermore, it provides fast convergence to the optimal solution. For b-instances, more than 80% of the optimal solution is found in less than 10% of the time required by DP.

In the CG context, MDDPA has two main advantages. First, as a primal method, it provides a sequence of columns of nonincreasing cost, and it allows premature stopping of the solution process when the quality of the columns is sufficient. Second, it offers two loading strategies, allowing different strategies to be used at different stages of the CG. Future research will focus on embedding the MDDPA within a CG scheme and using it in different applications.

Acknowledgements

This work was supported by a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Kronos Inc. The authors would like to thank all the members of these organizations for their support and their confidence.

CHAPITRE 6 ARTICLE 2 : A PRIMAL ADJACENCY-BASED ALGORITHM FOR THE SHORTEST PATH PROBLEM WITH RESOURCE CONSTRAINTS

Le texte de ce chapitre est celui de l'article (Himmich *et al.*, 2018b) :

A Primal Adjacency-Based Algorithm for the Shortest Path Problem with Resource Constraints

Ilyas Himmich, Hatem Ben Amor, Issmail El Hallaoui, François Soumis
GERAD et École Polytechnique de Montréal
Département de Mathématiques et Génie Industriel
C.P. 6079, Succ. Centre-Ville
Montreal, Quebec, Canada H3C 3A7
ilyas.himmich, hatem.ben.amor, issmail.elhallaoui, françois.soumis@gerad.ca

publié dans les cahiers du GERAD et soumis à

Transportation Science

date de soumission : Février 2018

Abstract

The shortest path problem with resource constraints (SPPRC) is often used as a subproblem within a column generation approach for routing and scheduling problems. It aims to find a least cost path between the source and the destination nodes in a network while satisfying the resource consumption limitations on every node. The SPPRC is usually solved using dynamic programming. Such approaches are effective in practice, but they can be inefficient when the network is large and especially when the number of resources is high. To cope with this major drawback, we propose a new exact primal algorithm that explores the solution space iteratively using a path-adjacency-based partition. Numerical experiments for vehicle and crew scheduling instances demonstrate that the new approach outperforms both the standard dynamic programming and the multi-directional dynamic programming methods.

Keywords : Shortest path problem with resource constraints, column generation, adjacency, dynamic programming.

6.1 Introduction

One of the most important problems in the context of large-scale transportation models is the shortest path problem with resource constraints (SPPRC). It was introduced by Desrochers (1986) as an extension of the shortest path problem with time windows. It aims to find the least cost path between the source and the destination nodes in a network, while satisfying the resource consumption limits on every node. The SPPRC is often used as a subproblem in the solution of vehicle routing and crew scheduling problems by column generation (CG). In the airline industry, solving the crew rostering problem by CG requires at each iteration the solution of hundreds or thousands of subproblems, one for each crew member. Resources are used to model the collective agreement and safety rules as well as other aspects related to the solution quality. A few dozen resources are generally used.

We propose in this work a new exact primal adjacency-based (PAB) algorithm that explores the solution space iteratively using a path-adjacency-based partition. Our study will focus on acyclic networks, which are often used in vehicle and crew scheduling applications in the airline industry (Desaulniers & Villeneuve, 2000; Nagih & Soumis, 2006).

6.1.1 Literature review on exact solution methods

The SPPRC has been widely studied, and several exact and heuristic algorithms have been proposed. They can be classified into three main groups (see Pugliese & Guerriero, 2013b) : 1) path ranking methods, 2) branch-and-bound (B&B) based methods, and 3) dynamic programming (DP) methods. Path ranking methods sort the paths in the network by increasing cost until a feasible one is found. The first path ranking algorithm was proposed by Handler & Zang (1980), and it was improved by Santos *et al.* (2007). The major drawback of this approach is that the number of paths that must be ranked increases exponentially with the size of the network. This makes the method computationally intractable.

B&B approaches have been proposed by Beasley & Christofides (1989), Carlyle *et al.* (2008), and Muhandiramge & Boland (2009). Each node in the B&B tree represents a subpath from the source node s to a given node in the network. The nodes for infeasible subpaths are removed from the tree, and upper and lower bounds are used to prune unpromising nodes. Thus, the efficiency depends on the quality of these bounds.

In the context of CG, DP has proved to be the most effective method in practice. The basic DP algorithm for a simplified version of the SPPRC is an extension of the Bellman–Ford algorithm (Desrochers & Soumis, 1988a). The algorithm assigns *labels* to each node i. A label is a multidimensional vector storing the cost and the total resource consumptions of a subpath from the source node s to a node i. Label dominance rules as well as pruning strategies are used to omit unpromising subpaths, i.e., subpaths that cannot be used to produce an optimal path. This approach can handle complex rules that may be nonlinear and even nonconvex, and it can provide the CG master problem with several columns at each iteration.

Several researchers have developed efficient DP algorithms for the SPPRC, e.g., Desrosiers *et al.* (1983); Desrochers & Soumis (1988a,b). An improved version of the labeling algorithm described in Desrochers & Soumis (1988a) was proposed by Dumitrescu & Boland (2003). They use information from the solution of a Lagrangean dual problem to compute lower and upper bounds. This prunes more labels, which reduces the solution time. Muhandiramge & Boland (2009) introduced preprocessing techniques to reduce the size of the network. New refinements of the solution of the SPPRC inside CG process, were proposed by Feillet *et al.* (2007). One of these refinements is a limited discrepancy search algorithm that prioritizes the most promising arcs when extending labels. Another exact solution algorithm was introduced by Lozano *et al.* (2016), it relies on implicit enumeration of feasible paths, and uses some pruning schemes to narrow the search space.

Recently, Himmich *et al.* (2018a) have proposed a new multi-directional dynamic programming algorithm (MDDPA) for the SPPRC. This approach is a generalization of the classical mono-directional DP algorithm. It stores, in an initialization step, sets of efficient labels at different nodes of the network, before extending these labels sequentially in several iterations according to a predetermined search strategy. MDDPA provides two search strategies : Nearest First first extends the labels closest to the destination node, and Best First prioritizes the extension of the labels with the most important costs (the most negative reduced costs in CG). This algorithm performed well compared to the standard DP method.

6.1.2 Approximate algorithms for column generation

Solving large instances of the SPPRC with DP is time-consuming, especially when the set of rules is large and complex. In fact, as the number of resources increases, the number of labels becomes excessive for large networks. Many approximate methods have been developed to cope with this major drawback. In a CG context, we need to solve the SPPRC to proven optimality only in the last iteration, to show that there are no more negative reduced cost paths. Approximate algorithms suffice in the preceding iterations. These techniques are based on limiting the number of labels to retain at a given node ; extending the labels on a subset of arcs, or testing dominance on a preselected subset of resources (Irnich & Desaulniers, 2005). These techniques drastically reduce the number of labels at each iteration, but the solution is likely suboptimal, and the process may be time-consuming. In addition, the choice of the labels to keep, the arcs to use, and the resources to dominate on is based on the intuition and experience of the planners rather than mathematical rules, so the algorithm requires the adjustment of several parameters for each problem and potentially each instance.

Nagih & Soumis (2006) proposed a different approach to mitigate the impact of the large number of resources. They project the resource vector onto a lower dimensional subspace. They define a Lagrangean dual problem by relaxing a subset of the resource constraints, and the corresponding Lagrangean multipliers are the coefficients of the projection matrix. The experiments show that their relaxation must be modified at each iteration to generate adequate negative reduced cost paths.

6.1.3 Contributions and organization

The contributions of this paper are fivefold :

i. We present a new polyhedral study that allows us to split the search space of the SPPRC into small disjoint subspaces. This partition is defined using path-adjacency-based nested neighborhoods.

ii. We develop a new primal exact algorithm that explores iteratively these subspaces using an improved version of DP without regeneration of labels, until optimality is proved.

iii. Unlike the existing methods, including MDDPA, the PAB algorithm benefits from available primal information, e.g., previous schedules, to build a good starting solution. This primal information is enriched at the subsequent iterations by the most promising paths generated during the solution process.

iv. Being a primal method, the PAB algorithm is able to produce sets of feasible paths of nonincreasing cost at each iteration. We use these paths for two purposes : First, to tighten the cost bounds at each node of the network, which reduces the size of the search space in the subsequent iterations. Second, to construct a good starting point for a subsequent iteration using affine combinations of the previously generated paths. This greatly reduces the number of labels generated and thus accelerates the solution process.

v. Our extensive experiments show that the PAB algorithm is better than the state-of-the-art DP algorithms on large-scale acyclic network instances with up to 600.000 nodes and 1.000.000 arcs. It returns very interesting solutions in very limited portions of time, and drastically reduces the number of created labels. These results show that the proposed approach provides a highly efficient solution framework, nicely suitable for CG method.

The rest of the paper is organized as follows. Section 6.2 formally defines the SPPRC using the cocycle-based formulation introduced by Himmich *et al.* (2018a), and Section 6.3 presents a polyhedral study where path-adjacency in networks is defined and linked to the notion of a *detour*. The new algorithm is presented in Section 6.4, and its efficiency is demonstrated in Section 6.5 where an extensive numerical study is discussed. Section 6.6 provides concluding remarks.

6.2 Mathematical formulation

Consider a connected acyclic network $G(V, A)$ where V is the set of nodes including the source node s and the destination node d, and $A = \{(i, j) \mid i, j \in V\}$ is the set of arcs. The SPPRC computes the minimum-cost path among all feasible paths starting from s and ending at d. A path is said to be feasible if it respects the resource constraints induced by a set of resources $\mathcal{R}$.

The resource constraints are based on resource consumptions and resource intervals. The resource consumptions are $|\mathcal{R}|$-dimensional vectors $(r_{ij}^1, r_{ij}^2, ..., r_{ij}^{|\mathcal{R}|})$ associated with each arc

$(i,j) \in A$ where r_{ij}^t is the quantity of resource $t \in \mathcal{R}$ consumed when traversing arc (i,j). The resource intervals, also called resource windows, are a set of intervals $[a_i^t, b_i^t]$ associated with each node $i \in V$, where a_i^t and b_i^t are the lower and upper bounds on the resource $t \in \mathcal{R}$ consumed along an *s-i* subpath.

DP algorithms associate with each *s-i* subpath π_i an $(|\mathcal{R}|+1)$ vector $(C_i, R_i^1, R_i^2, ..., R_i^{|\mathcal{R}|})$ called a *label*. C_i and R_i^t denote respectively the cost and the consumption of each resource $t \in \mathcal{R}$ over all the arcs composing π_i. In most applications, π_i is feasible if $a_i^t \leq R_i^t \leq b_i^t$, $\forall t \in \mathcal{R}$, however, it may be allowed that $R_i^t < a_i^t$ as in the case of time windows.

In addition to the resource constraints, path-structure constraints are used to ensure the connectivity of the path. These constraints are basically flow conservation constraints that allow the circulation of one unit of flow from s to d. Additional path-structure constraints may be added to the formulation depending on the requirements of the problem ; there may be forbidden paths, or a need for elementary paths in cyclic networks. Let x_{ij} be the arc binary variable that takes the value 1 if the arc $(i,j) \in A$ is chosen to be part of the solution, 0 otherwise. The SPPRC is formulated as follows :

$$(P_1) \quad \textit{Minimize} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{6.1}$$

s.t.

$$\sum_{i\in V} x_{ij} - \sum_{i\in V} x_{ji} = \begin{cases} -1 & \text{for } j = s \\ 0 & \forall\ j \in V \setminus \{s,d\} \\ 1 & \text{for } j = d \end{cases} \tag{6.2}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \quad \forall t \in \mathcal{R}, \forall (i,j) \in A \tag{6.3}$$

$$a_i^t \leq R_i^t \leq b_i^t \quad \forall t \in \mathcal{R}, \forall i \in V \tag{6.4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{6.5}$$

Constraints (6.2) are the flow conservation constraints. Constraints (6.3) model the resource consumptions whenever an arc (i,j) is chosen to be part of the solution (path), while constraints (6.4) require the resource consumption along each *s-i* subpath to be within the corresponding resource interval. Note that if it is allowed to arrive at a node $i \in V$ even if $R_i^t < a_i^t$ for some $t \in \mathcal{R}$; the resource consumption R_i^t takes the value a_i^t. Constraints (6.5) are the binary requirements on the arc variables x_{ij}, $(i,j) \in A$. We note that the previous

model assumes the linearity of the cost and the resources' extension functions. In practice, these functions may be nonlinear or even non-convex.

The flow conservation constraints (6.2) were reformulated by Himmich *et al.* (2018a) using the notion of a *cocycle*. They first sort the nodes of $G(V, A)$ in topological order. Each node is denoted by its rank in that order, and the source and destination nodes are indexed respectively by 1 and $|V|$. A cocycle is defined as follows.

Definition 8. *(Himmich et al., 2018a) Let $G(V, A)$ be an acyclic network, where $V = \{1, 2, ..., |V|\}$ is topologically ordered, and consider a node $k \in V$. The cocycle k, denoted Co_k, is the set of all arcs $(i, j) \in A$ such that the origin $i \in V$ is ordered before node k and the destination j is ordered strictly after node k. Formally, $Co_k = \{(i, j) \in A | i \leq k < j\}$.*

The authors associate with each cocycle a constraint called a *cocycle constraint*. These constraints are as follows :

$$\sum_{(i,j) \in Co_k} x_{ij} = 1 \qquad \forall k \in \{1, 2, ..., |V| - 1\}. \tag{6.6}$$

Figure 6.1, reproduced from (Himmich *et al.*, 2018a), shows the cocycle constraints for a four-node acyclic network.

Figure 6.1 Cocycle constraints

Himmich *et al.* (2018a) proved that the cocycle constraints are equivalent to the flow conservation constraints in acyclic connected networks. Hence, they reformulate the SPPRC as a set partitioning problem with side constraints where each column represents an arc covering a subset of the cocycles. A solution to the problem is a feasible path covering all cocycles exactly once. The new formulation is as follows :

$$
\begin{aligned}
(P_2) \quad & \textit{Minimize} \quad \boldsymbol{c}^T \boldsymbol{x} \\
\textit{s.t.} \quad & \boldsymbol{M}\boldsymbol{x} = \boldsymbol{e} \\
& (6.3) - (6.5)
\end{aligned}
$$

where $\boldsymbol{M} = [m_{ka}]_{(|V|-1).|A|}$ such that $k \in V \setminus \{d\}, a \in A$ is a $(|V|-1) \times |A|$ matrix such that $m_{ka} = 1$ if arc a covers cocycle Co_k and 0 otherwise; $\boldsymbol{c}$ is the arc cost vector; and $\boldsymbol{e} = (1, ..., 1)^T$ is a (|V|-1) vector.

Proposition 7. *The matrix $\boldsymbol{M}$ is full rank.*

Proof. Since the network G is connected, for each cocycle Co_k, there is at least one arc a such that $m_{ka} = 1$ and $m_{la} = 0 \ \forall l < k$. The columns corresponding to these arcs can then be rearranged to form a triangular submatrix $\boldsymbol{N}$ such that $\boldsymbol{N}_{kk} = 1$ and $\boldsymbol{N}_{kl} = 0$, $\forall k \in \{1, 2, ..., |V|-1\}, l < k$. Consequently, the $|V|-1$ rows are linearly independent, which completes the proof. □

In what follows, we denote by $\mathcal{P}^{SPP}$ the polyhedron for the ordinary shortest path problem (obtained by relaxing the resource constraints (6.3) and (6.4) and the integrality constraints (6.5)).

Corollary 1. *Since the $|V|-1$ cocycle constraints are linearly independent, the dimension of $\mathcal{P}^{SPP}$ is $|A| - |V| + 1$.*

6.3 Polyhedral study

We now study some polyhedral properties related to path adjacency in $\mathcal{P}^{SPP}$. We discuss adjacency to a single path in Section 6.3.1, and we generalize this to a set of paths in Section 6.3.2. Based on the path adjacency, we derive the notion of degree of adjacency, which we will use to classify the feasible solutions to SPPRC. This classification is used to partition the solution space of the SPPRC into several disjoint subspaces that will be explored by the PAB algorithm (in the next section).

We decided to focus on $\mathcal{P}^{SPP}$ instead of the solution space of the SPPRC for three reasons. First, all the solutions of the SPPRC are paths from s to d, and their corresponding extreme points are all integers. Therefore, every feasible extreme point in the solution space of the SPPRC is also an extreme point of $\mathcal{P}^{SPP}$. Second, the adjacency between two feasible extreme points in $\mathcal{P}^{SPP}$ remains valid in the convex hull of the solutions of SPPRC. Third, every

partition of the set of solutions in $\mathcal{P}^{SPP}$ induces a partition in the solution space of the SPPRC. Thus, if we consider in addition that the solution space of the SPPRC is not convex in general, and does not define a polyhedron, we assume that $\mathcal{P}^{SPP}$ allows the extraction of polyhedral properties describing the solutions of SPPRC. Obviously, $\mathcal{P}^{SPP}$ has been widely studied in both graph theory and linear optimization, but to the best of our knowledge, the results we present are new, and this is the first work to use the notion of adjacency to solve the SPPRC.

6.3.1 Adjacency to a single path

As seen in Section 6.2, the ordinary shortest path problem can be equivalently formulated as the following set partitioning problem :

$$\begin{aligned} (P_3) \quad & \textit{Minimize} \quad \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & \boldsymbol{M}\boldsymbol{x} = \boldsymbol{e} \\ & \boldsymbol{x_a} \in \{0,1\} \quad a \in A \end{aligned}$$

where x_a denotes the variable associated with arc $a \in A$ arc, and $\boldsymbol{M_a}$ its related column. We consider a basic integer solution $\bar{\boldsymbol{x}}$ to (P_3), and we denote by P the index set of all the positive-valued basic variables of $\bar{\boldsymbol{x}}$. The cocycle formulation allows the use of the compatibility defined by Zaghrouti *et al.* (2014) for set partitioning problems.

Definition 9. *A nonempty set of columns $\mathcal{D}$ is said to be compatible with the basic integer solution $\bar{\boldsymbol{x}}$ (or simply compatible) if there is a subset of columns with indices in $P' \subseteq P$ such that $\sum_{a\in\mathcal{D}} \boldsymbol{M_a} = \sum_{a\in P'} \boldsymbol{M_a}$. The set of variables corresponding to columns of $\mathcal{D}$ is also said to be compatible.*

The set of arcs composing a path π from s to d is denoted A^π, and the corresponding vector $\boldsymbol{x^\pi} \in \boldsymbol{\{0,1\}^{|A|}}$ is such that $x_a^\pi = 1$ *iff* arc $a \in A^\pi$. Recall that in the shortest path problem, a basic solution corresponds to a spanning tree with the source node as the origin. This solution consists of sending one unit of flow along the unique path π in the tree going from s to d. Consequently, the positive-valued variables correspond to the arcs $a \in A^\pi$, while the zero-valued variables correspond to the rest of the arcs. Furthermore, a compatible set of variables corresponds to a set of arcs in $A \setminus A^\pi$ that can replace (a subset of) A^π to form a new path. Using the notion of a cocycle introduced in Definition 8, we may view a set of compatible columns as a set of arcs that cover the same cocycles as those covered by a subset of the arcs in A^π. The next definition tightens the notion of compatibility using the minimality of a set of compatible columns.

Definition 10. *Let π be a path corresponding to a given basic solution $\bar{x}$. A set of arcs $\mathcal{D}$ is called a detour if it is compatible with $\bar{x}$ and minimal, in the sense that none of its strict subsets is compatible.*

A detour may also be seen as a subpath with exactly one arc leaving π and one arc entering it, as illustrated in Figure 6.2.

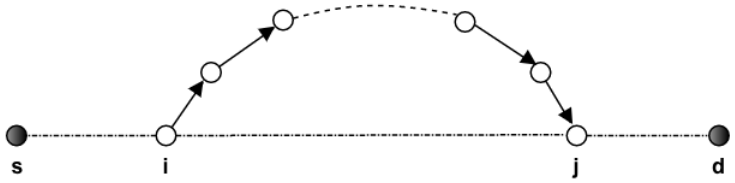

Figure 6.2 Notion of detour

We now recall the definition of adjacency in linear programming.

Definition 11. *Let $\mathcal{P}$ be the polyhedron of a linear program. Two extreme points $\boldsymbol{x^1}$ and $\boldsymbol{x^2}$ of $\mathcal{P}$ are adjacent if there exists a face of $\mathcal{P}$ of dimension 1 (an edge) that contains both $\boldsymbol{x^1}$ and $\boldsymbol{x^2}$.*

The following proposition establishes the link between the notions of detour and path adjacency in $\mathcal{P}^{SPP}$.

Proposition 8. *A path $\pi^{'}$ is adjacent to π iff there exists exactly one detour $\mathcal{D}$ associated to π such that $A^{\pi^{'}} \setminus A^{\pi} = \mathcal{D}$.*

Proof. $\Leftarrow$ Consider a detour $\mathcal{D}$ such that $A^{\pi^{'}} \setminus A^{\pi} = \mathcal{D}$, and let F be the face of least dimension containing x^{π} and $x\pi^{'}$. $F = \{\boldsymbol{x} : \boldsymbol{M}\boldsymbol{x} = \boldsymbol{e}\ ;\ x_a = 1\ \forall a \in A^{\pi} \cap A^{\pi^{'}};\ x_a = 0\ \forall a \in A \setminus (A^{\pi} \cup A^{\pi^{'}})\}$. On the one hand, $\boldsymbol{x_{\pi}}$ and $\boldsymbol{x_{\pi'}}$ are affinely independent, so $dim(F) \geq 1$ (1). On the other hand, if we denote by $F^{=}$ the matrix of the equality constraints satisfied by F, then $dim(F) = |A| - rank(F^{=})$.

Consider the subnetwork $G^*(V^*, A^*)$ composed of only the arcs in $(A^{\pi} \cup A^{\pi^{'}}) \setminus (A^{\pi} \cap A^{\pi^{'}})$ and their origine and destination nodes. We have $|V^*| = |A^*| = |(A^{\pi} \cup A^{\pi^{'}}) \setminus (A^{\pi} \cap A^{\pi^{'}})|$. The number of cocycle equalities in G^* is then $|V^*| - 1 = |(A^{\pi} \cup A^{\pi^{'}}) \setminus (A^{\pi} \cap A^{\pi^{'}})| - 1$. As a result, $F^{=}$ is defined by at least $|A^{\pi} \cap A^{\pi^{'}}| + |(A^{\pi} \cup A^{\pi^{'}}) \setminus (A^{\pi} \cap A^{\pi^{'}})| + |A \setminus (A^{\pi} \cup A^{\pi^{'}})| - 1 = |A| - 1$ equalities that are linearly independent. Therefore, $rank(F^{=}) \geq |A| - 1$, which implies

$dim(F) = |A| - rank(F^{=}) \leq |A| - |A| + 1 = 1$ (2). By (1) and (2), $dim(F) = 1$, so π_1 and π_2 are adjacent in $\mathcal{P}^{SPP}$.

$\Rightarrow$ Suppose that π and $\pi^{'}$ are two adjacent paths, and there are $k > 1$ detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^{\pi^{'}} \setminus A^{\pi} = \cup_{i=1}^{k} \mathcal{D}_i$. Let F be the face of degree 1 containing $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Consider the sequence of paths π_i such that $A^{\pi_i} \setminus A^{\pi} = \mathcal{D}_i$, $i = \{1, 2, ..., k\}$. This implies $A^{\pi^{'}} \setminus A^{\pi} = \cup_{i=1}^{k} (A^{\pi_i} \setminus A^{\pi})$. Thus, $\boldsymbol{x_{\pi'}} - \boldsymbol{x_\pi} = \sum_i (\boldsymbol{x_{\pi_i}} - \boldsymbol{x_\pi})$, so $\boldsymbol{x_{\pi'}}$ is a linear combination of the set of $(k+1)$ extreme points $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_k}}\}$. Therefore, these extreme points are contained in F. In addition, since they are affinely independent, we have $dim(F) \geq k > 1$. Hence, we have a contradiction. □

Let Adj_π be the set of all paths adjacent to π. The notion of adjacency can be extended to the general case where two extreme points are contained in a face of dimension $k > 1$. We define the degree of adjacency as follows :

Definition 12. *In $\mathcal{P}^{SPP}$, the degree of adjacency of two paths π and $\pi^{'}$ is equal to the least dimension k of a face containing $\boldsymbol{x^\pi}$ and $\boldsymbol{x^{\pi'}}$. We say that π and $\pi^{'}$ are k-adjacent.*

The next proposition gives a generalization of Proposition 8.

Proposition 9. *A path $\pi^{'}$ is k-adjacent to a path π iff there exist exactly k detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^{\pi^{'}} \setminus A^{\pi} = \cup_{i=1}^{k} \mathcal{D}_i$.*

Proof. Let G^* be the subnetwork composed of only the arcs in $A^\pi \cup A^{\pi^{'}}$.

$\Leftarrow$ Assume that there are k detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^{\pi^{'}} \setminus A^{\pi} = \cup_{i=1}^{k} \mathcal{D}_i$. Let $\{\pi_1, \pi_2, ..., \pi_k\}$ be the set of all paths π_i adjacent to π in G^* such that $A^{\pi_i} \setminus A^{\pi} = \mathcal{D}_i \ \forall i$, and let F be the face of the least dimension containing x_π and x_{π_i} for $i = \{1, 2, ..., k\}$. On the one hand, the set $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_k}}\}$ is affinely independent, so $dim(F) \geq k$. On the other hand, as in the previous proof, we can easily show that the rank of the equalities matrix $F^{=}$ is $\geq (|A| - 1)$, which leads us to conclude that $dim(F) = |A| - rank(F^{=}) \leq k$. Hence, $dim(F) = k$.

$\Rightarrow$ π and $\pi^{'}$ are k-adjacent means, by definition 12, that k is the dimension of the face F of least dimension containing $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Suppose that $d < k$ is the number of detours between $\boldsymbol{x_\pi}$ and $\boldsymbol{x_{\pi'}}$. Then there exists a face of dimension d containing π and $\pi^{'}$, and F is no longer the face of least dimension. Suppose now that $d > k$. Since F contains π and $\pi^{'}$, it will contain the $(d+1)$ affinely independent extreme points $\{\boldsymbol{x_\pi}, \boldsymbol{x_{\pi_1}}, \boldsymbol{x_{\pi_2}}, ..., \boldsymbol{x_{\pi_d}}\}$ defined as follows : $\pi_1 \in Adj_\pi$, $\pi_i \in Adj_{\pi_{i-1}}$ for $i \in \{2, ..., d\}$ and $\boldsymbol{x_{\pi_d}} = \boldsymbol{x_{\pi'}}$. Thus, $dim(F) \geq d > k$. Hence, we have a contradiction. □

The last proposition can be interpreted as the existence of a sequence of k edges in $\mathcal{P}^{SPP}$ linking $\boldsymbol{x_\pi}$ to $\boldsymbol{x_{\pi'}}$. In addition, the intermediate extreme points correspond to paths that are constructed of arcs and nodes of π and π'. The set of all paths k-adjacent to π is denoted Adj_π^k.

Corollary 2. *The sets of paths Adj_π^k for $k \in \{1, 2, ..., |A^\pi|\}$ form a partition of the set of all paths.*

Proof. Since the degree of adjacency is well defined (unique), the sets Adj_π^k are disjoint. Moreover, the partition size is upper bounded by the number of arcs $|A^\pi|$, since the maximum degree of adjacency is upper bounded by $|A^\pi|$. □

6.3.2 Adjacency to a set of paths

We now investigate the notion of adjacency of a path π to a set of paths Π (all from s to d). We denote by $A^\Pi = \cup_{\pi\in\Pi} A^\pi$ the set of arcs and by V^Π the set of nodes. We denote by $\mathcal{S}$ the set of all paths, while S^Π indicates the subset of $\mathcal{S}$ containing the paths composed of arcs of A^Π.

Definition 13. *Let P be the index set of all arcs in Π. A set of arcs $\mathcal{D}$ is said to be compatible with Π iff there are two subsets $P^+, P^- \subset P$ such that $\sum_{a\in\mathcal{D}} \boldsymbol{M_a} = \sum_{a\in P^+} \boldsymbol{M_a} - \sum_{a\in P^-} \boldsymbol{M_a}$. If in addition $\mathcal{D}$ is minimal, we call it a detour.*

Remark 3. *Similarly to the case of adjacency to a single path, a detour corresponds to a subpath containing exactly one arc leaving Π and one arc entering it, as shown in Figure 6.3.*

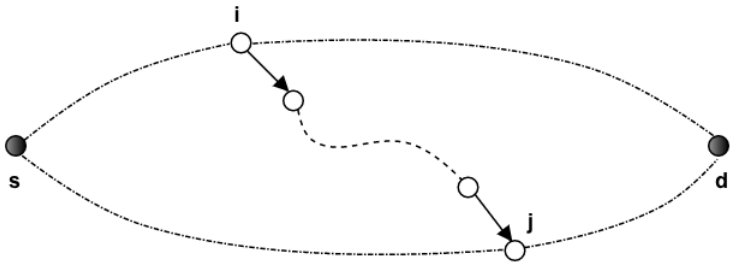

Figure 6.3 Detour from i to j for a set of two paths

Definition 14. *A path π is k-adjacent to Π if there are exactly k detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^\pi \setminus A^\Pi = \cup_{i=1}^k \mathcal{D}_i$. We write $\pi \in Adj_\Pi^k$.*

Proposition 10. *If a path π is k-adjacent to Π, then there is at least one path π_0 in S^{Π} and $1 \leq t \leq k$ such that $\pi \in Adj^t_{\pi_0}$.*

Proof. $\pi \in Adj^k_{\Pi}$ implies the existence of k detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^{\pi} \setminus A^{\Pi} = \cup_{i=1}^{k} \mathcal{D}_i$. Let π_0 be a path of S^{Π} that shares the maximum number of arcs with π. The number of detours made by π in relation to π_0 is upper bounded by k, so $\pi \in Adj^t_{\pi_0}$ while $t \leq k$. Moreover, it is obvious that $t \geq 1$ since $k \geq 1$. □

Consider a path π that is $k-$adjacent to a set of paths Π. We show below that there is an affine basis generating S^{Π} such that π is at most $(k+1)-$adjacent to every path in the basis. We need the following preliminary result :

Proposition 11. *Consider a path π_0 and a sequence of sets Π_k such that $\Pi_0 = \{\pi_0\}$ and $\Pi_i = \Pi_{i-1} \cup \{\pi_i\}$ where $\pi_i \in Adj^1_{\Pi_{i-1}}$, $i \in \{1, 2, ..., |A| - |V| + 1\}$. The set $\mathcal{B} = \{\boldsymbol{x}_{\pi_i}, i = 0, 1, ..., |A| - |V| + 1\}$ is an affine basis of $\mathcal{P}^{SPP}$.*

The next lemma is useful for proving this result.

Lemma 1. *The dimension of $\mathcal{P}^{SPP}$ is equal to the number of detours needed to cover the whole network starting with an initial path.*

Proof. The proof is by induction on the dimension of the polyhedron. For a 0-dimensional polyhedron, there is exactly one path in the network and 0 detours, and the lemma holds. Let $G^n(V^n, A^n)$ be an acyclic network and $\mathcal{P}^n$ the polyhedron corresponding to G^n. By Corollary 1, we have $dim(\mathcal{P}^n) = |A^n| - |V^n| + 1$. Suppose that $dim(\mathcal{P}^n) = n$ where n is the number of detours needed to cover G^n. We add to the network G^n one detour $\mathcal{D}$ with a number of arcs $|\mathcal{D}|$. The number of newly added nodes is then equal to $|\mathcal{D}| - 1$. We call the resulting network $G^{n+1}(V^{n+1}, A^{n+1})$, and its corresponding polyhedron is $\mathcal{P}^{n+1}$. There are $|V^{n+1}| - 1 = |V| + |\mathcal{D}| - 1$ linearly independent cocycle equality constraints. Thus, $dim(\mathcal{P}^{n+1}) = |A^{n+1}| - |V^{n+1}| + 1 = |A| + |\mathcal{D}| - (|V| + |\mathcal{D}| - 1) + 1 = |A| - |V| + 2 = dim(\mathcal{P}^n) + 1 = n + 1$. This completes the proof. □

Proof of Proposition 11. By Lemma 1, there are $|A| - |V| + 1$ detours needed to cover the whole network, where each detour generates a path π_i that is affinely independent of the set of paths Π_{i-1}. This gives a total of $|A| - |V| + 1$ affinely independent paths. When we add the initial path π_0, the set $\{\boldsymbol{x}_{\pi_i}, i = 0, 1, ..., |A| - |V| + 1\}$ is affinely independent and generates $\mathcal{S}$, so it forms an affine basis of $\mathcal{P}^{SPP}$. □

Proposition 12. *Let π be a k-adjacent path to Π. There is an affine basis $\mathcal{B} = \{\boldsymbol{x}_{\boldsymbol{\pi_i}}\}_{(i=0,1,...,n)}$ generating S^{Π} such that $\forall i \in \{0, 1, ..., n\}$, $\exists$ t such that $1 \leq t \leq k + 1$, $\pi \in Adj^t_{\pi_i}$.*

Proof. Consider $\pi \in Adj^k_\Pi$. There is at least one path in $S^\Pi \cap Adj^k_\pi$; let π_0 be one of these paths. We choose a series of paths $\pi_i \in S^\Pi$ such that $\pi_i \in Adj^1_{\Pi_{i-1}} \cap Adj^1_{\pi_0}$, $\Pi_0 = \{\pi_0\}$, $\Pi_i = \Pi_{i-1} \cup \{\pi_i\}$, and $\Pi_n = \Pi$. Since $\pi \in Adj^k_{\pi_0}$, π is at most $(k+1)$-adjacent to π_i $\forall i$. Moreover, from Proposition 11 the set $\{\boldsymbol{x_{\pi_0}}, \boldsymbol{x_{\pi_1}}, ..., \boldsymbol{x_{\pi_n}}\}$ is affinely independent and generates S^Π. $\mathcal{B}$ is then an affine basis generating S^Π. □

Remark 4. *Let* Π *be a set of paths and* $\mathcal{B}$ *the affine basis generating* S^Π*. We note that every path in* S^Π *that is a combination of the elements of the basis* $\mathcal{B}$ *is 0-adjacent to* Π*. These paths will be generated by the Combination step of the PAB algorithm that will be introduced in Section 6.4.*

Proposition 13. *Consider a set of paths* Π *from s to d, the set of paths* S^Π *belong to a face of* $\mathcal{P}^{SPP}$*.*

Proof. Let $G^\Pi(A^\Pi, V^\Pi)$ be a subnetwork of $G(V, A)$ containing of all the nodes of arcs composing Π. The dimension of the polyhedron corresponding to subnetwork G^Π is $|A^\Pi| - |V^\Pi| + 1$. According to Proposition 11, we can extract exactly $|A^\Pi| - |V^\Pi| + 2$ affinely independent paths from S^Π. Consequently, this set of paths generates a face of $\mathcal{P}^{SPP}$, with dimension $|A^\Pi| - |V^\Pi| + 1$. □

Proposition 14. *Let* $\mathcal{B}$ *be a basis of the polyhedron corresponding to a network* $G(V, A)$*, every path from s to d in* $G(V, A)$ *is an affine combination of the elements of* $\mathcal{B}$*.*

Proof. Let π be a path in $G(V, A)$. $\mathcal{B} = \{\boldsymbol{x_{\pi_i}}, \boldsymbol{i = 0, 1, ..., |A| - |V| + 1}\}$ is a basis. Thus, there exists α_i such that $x^\pi = \sum_i \alpha_i \boldsymbol{x^{\pi_i}}$. Moreover, we know that every path satisfies the cocycle constraints. Thus, $\boldsymbol{M x^\pi} = \sum_i \alpha_i \boldsymbol{M x^{\pi_i}} = (\sum_i \alpha_i)\boldsymbol{e} = \boldsymbol{e}$, which implies that $\sum_i \alpha_i = 1$. □

Corollary 3. *The sets of paths* Adj^k_Π *for* $k \in \{1, 2, ..., k_{max}\}$ *form a partition of* $\mathcal{S}$*, where* k_{max} *is an upper bound on the degree of adjacency to* Π*.*

6.4 Primal adjacency-based algorithm

Globally, the PAB algorithm (hereafter referred to as PAB), is the result of the combination of DP and the polyhedral concepts and results introduced in the previous section. The main features of PAB are the following : first, instead of executing one search in the whole network, the algorithm performs a DP sequence in limited search spaces. This search space reduction allows the algorithm to explore larger networks. Second, the algorithm can use the specific

structure of the application networks to construct an initial set of paths rich in primal information. This set is dynamically improved at each iteration until optimality is reached. Third, thanks to the combination property introduced in Proposition 14, the algorithm can combine the generated paths to produce good improving paths in a limited computational time.

We introduce the PAB pseudocode and describe its main steps in Section 6.4.1. We then present in Section 6.4.2 an improved labeling algorithm (ILA) that is the main component of PAB. Finally, we discuss its convergence in Section 6.4.3. We introduce the following notation.

— k : Iteration/degree of adjacency.

— k_{max} : Maximum degree of adjacency. This degree is defined according to the initial point as described in Section 6.5.1.3.

— C^{best} : Cost of the current best feasible path.

— Π : Current set of generated paths.

— C_{di} : Cost of the reverse shortest path from $i \in V$ to the destination d.

— U_i : Cost upper bound for node $i \in V$.

— $\mathcal{L}_i$: Active labels in node i. ($\mathcal{L} = \cup_{i \in V} \mathcal{L}_i$).

— $\mathcal{S}_i^k$: Saved labels of degree k in node i. ($\mathcal{S}_i = \cup_{k=1}^{k_{max}} \mathcal{S}_i^k$; $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$).

— Ω^c, Ω^e : Set of Pareto-optimal paths provided by *Combination* and *Extension* steps, respectively. ($\Omega = \Omega^c \cup \Omega^e$).

6.4.1 PAB pseudocode

Starting from an initial set of paths $\Pi = \Pi_0$, the algorithm iteratively explores the elements of a partition of the feasible domain using the degree of adjacency as a measure of the distance to Π_0. This measure provides a partition of the solution space, as mentioned in Corollary 3. The search proceeds from lower degrees of adjacency to higher degrees. At the end of each iteration the promising paths found are added to Π. These paths are used for two purposes. First, they are combined in the *Combination* step to generate improving paths with higher degrees of adjacency ; second, their best cost helps to tighten the cost upper bounds on every node, allowing the elimination of many useless labels. This drastically reduces the computational complexity of the subsequent iterations. In Algorithm 6.1, we give the PAB pseudocode and describe its four main steps.

Algorithm 6.1: Primal adjacency-based algorithm

Initialization step
Find an initial set of paths Π_0 and compute $C_{di}, i \in V$.
$k \leftarrow 1$; $\mathcal{S}_s \leftarrow \{(0, ..., 0)\}$, $\mathcal{S}_i \leftarrow \emptyset, \forall i \neq s$; $\Pi \leftarrow \Pi_0$.
Combination step
$\mathcal{L}_i \leftarrow \mathcal{S}_i$ for all $i \in V^{\Pi}$.
$\Omega^c \leftarrow ILA(k_{max}, G^{\Pi}, \mathcal{L}, \mathcal{S})$.
Extension step
$\mathcal{L}_i \leftarrow \mathcal{S}_i^k$ for all $i \in V$.
$\Omega^e \leftarrow ILA(k, G, \mathcal{L}, \mathcal{S})$.
Control step
if $k = k_{max}$ **then**
  Stop : the solution is optimal ; return Π.
$k \leftarrow k + 1$.
$\Pi \leftarrow \Pi \cup \Omega$.
if $\Omega^e \neq \emptyset$ **then**
  go to **Combination step**.
else
  go to **Extension step**.

As mentioned earlier, a subpath π_i in node i is characterized by a label $l_i = (C_{l_i}, R^1_{l_i}, R^2_{l_i}, ..., R^{|\mathcal{R}|}_{l_i})$, which is an $(|\mathcal{R}|+1)$ vector with values representing the cost and resource consumption up to node i. A new *adjacency* resource $R^{adj}_{l_i}$ is added to the label definition to count the degree of adjacency to the path set Π_0. This resource counts the number of detours needed to construct π ; its upper bound is set to k for all the nodes. Based on Definition 14, it simply counts the number of arcs in path π that leave the set Π_0, and hence the corresponding arc consumption is 1 on the arcs leaving Π_0 and 0 otherwise. An arc $(i, j) \in A$ is said to be leaving the set Π_0 if $i \in V^{\Pi_0}$ and $(i, j) \notin A^{\Pi_0}$. Note that the new adjacency resource is used only to limit the search space ; it does not contribute to the dominance rules.

Initialization step. In this step, we initialize Π with a path set Π_0 that contains some good primal information. For crew scheduling problems, primal information is generally available : it can be extracted from previous schedules or taken from the predetermined bus lines or aircraft routes. We also compute $C_{di}, i \in V$ (by finding a reverse shortest path from d to s) for use in the *Combination* and *Extension* steps to tighten the upper bounds on the costs for all the nodes of the network. This preprocessing technique is well known in the literature, it was recently used by Righini & Salani (2006) in their bidirectional DP algorithm.

Combination step. Given a set of paths Π, this step seeks improving paths in S^{Π}. These paths are affine combinations of the elements of a basis of the polyhedron corresponding to

$G^{\Pi}(V^{\Pi}, A^{\Pi})$, as described in Proposition 14. Note that there is no need to build this basis in advance. Technically speaking, the *Combination* step consists of running ILA on $G^{\Pi}(V^{\Pi}, A^{\Pi})$, with a degree of adjacency to Π_0 equal to k_{max}. This means that the adjacency resource is not active, and ILA is allowed to construct improving paths using all the nodes and arcs in G^{Π} without any restriction. From a polyhedral point of view, this step may be viewed as a 0-*adjacency* search on the smallest face F containing the extreme points corresponding to the paths of Π. It searches for extreme points contained in F whose corresponding paths have not been generated yet. These paths generally have a higher degree of adjacency to Π_0 compared to the previously found paths that compose Π.

Moreover, since this step is performed in a small restricted subgraph G^{Π} of G, the set of labels corresponding to subpaths leaving G^{Π} are saved in $\mathcal{S}$ in order to be extended in the subsequent iteration corresponding to their degree of adjacency. In practice, this step is relatively fast, and it achieves significant cost improvements, as our experimental results will demonstrate. It is efficient because it considers relatively small subgraphs with primal information and because it generates improving paths of higher degrees of adjacency even before PAB reaches these degrees.

Extension step. This step extends the search space by increasing the degree of adjacency. We search here for improving paths having a higher adjacency degree when no such solutions are found in lower degrees. The search is performed by running ILA on the original network G, with a degree of adjacency k and a set of saved labels $\mathcal{S}$ from the previous iteration. Note that all the newly saved labels are of degree $k+1$. An updated set $\mathcal{S}$ of saved labels to be used in the subsequent iteration is produced.

Control step. If the maximum degree is reached, we assert that the current solution is optimal and stop the algorithm. Otherwise, if feasible paths are found for degree of adjacency k, they are added to the set of paths Π. This is done by adding to A^{Π} all the arcs for the detours of the paths. In this case, a new *Combination* step is carried out in the updated set Π. Otherwise, the algorithm enlarges the search space to the next degree of adjacency and performs a new search in the corresponding neighborhood.

From a polyhedral point of view, PAB looks for a better solution among the extreme points with a given degree of adjacency in relation to Π_0, and it increases this degree. By Proposition 13, these sets of paths define a face F_0 in $\mathcal{P}^{SPP}$. PAB enlarges the basis generating F_0 at each iteration by adding to it a set of newly generated paths, which creates a new face of larger dimension containing F_0. Finally, note that PAB does not need to find all the $|A| - |V| + 1$ elements of the basis generating S to prove optimality ; it adds only the most promising ones.

The reason is that PAB is able to produce paths having different degrees of adjacency to Π_0 (≥ 1) and not only those of degree 1. These k-adjacent paths contain the most promising detours that may be used to produce improving paths using affine combinations. We recall that a path π is k-adjacent to a set of paths Π if there exists a basis $\mathcal{B}$ generating Π such that every path in $\mathcal{B}$ is at most $(k+1)$-adjacent to π (Proposition 11).

6.4.2 Improved Labeling Algorithm

Recall that a labeling algorithm for SPPRC is an extension of the classical Bellman algorithm for SPP (Irnich & Desaulniers, 2005). The algorithm starts from a trivial subpath containing only the source node s and iteratively constructs new paths by extending available subpaths one-by-one in every direction satisfying the resource constraints imposed on the nodes. ILA is an improved labeling algorithm. It allows the generation of a set of feasible paths for a given degree of adjacency in an acyclic network $G(V, A)$ that is assumed to be topologically ordered. Given an adjacency degree k, a working graph $\bar{G}(\bar{V}, \bar{A})$, a set of active labels $\mathcal{L}_i$, and a set of saved labels $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$, $ILA(k, \bar{G}, \mathcal{L}, \mathcal{S})$ returns a set of Pareto-optimal paths Ω that are at most k degrees distant from Π_0. It also saves in $\mathcal{S}$ all the labels for efficient subpaths with higher degrees, or those that cannot be extended in $\bar{G}$, to avoid regenerating them in the subsequent iterations. The ILA pseudocode is given in Algorithm 6.2.

Algorithm 6.2: Improved Labeling Algorithm $ILA(k, \bar{G}, \mathcal{L}, \mathcal{S})$

1: **for all** $i \in V$ **do**
2: Update cost upper bound : $U_i \leftarrow C^{best} - C_{di}$.
3: **for all** $i \in \bar{V} \setminus \{d\}$ such that $\mathcal{L}_i \neq \emptyset$ (in increasing order) **do**
4: Delete dominated labels from $\mathcal{L}_i$.
5: **for all** $j \in V$ successor of i **do**
6: **for all** $l_i \in \mathcal{L}_i$ **do**
7: $l_j \leftarrow Extend(l_i, j)$.
8: **if** $j \in \bar{V}$ and l_j is feasible **then**
9: $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{l_j\}$.
10: **else**
11: **if** $j \notin \bar{V}$ or $k < R^{adj}_{l_j} \leq k_{max}$ (only resource constraint violated) **then**
12: $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{l_j\}$.
13: Build Ω from $\mathcal{L}_d$ and update C^{best}.
14: **Return** Ω.

In Step 2, we tighten the cost upper bound on the nodes to fathom the unpromising labels. The new upper bound at node i is obtained by subtracting C_{di}, the cost of the shortest path from i to the destination node d, from the best current cost C^{best}. Therefore, a sequence of nested sets of paths of nonincreasing cost is generated during the PAB process. This helps

to further reduce the number of labels generated at each iteration, already reduced by the neighborhood partition w.r.t. the degree of adjacency.

In Step 4, we remove dominated labels. A label $l = (C_l, R_l^1, R_l^2, ..., R_l^{|\mathcal{R}|}, R_l^{adj})$ dominates $l' = (C_{l'}, R_{l'}^1, R_{l'}^2, ..., R_{l'}^{|\mathcal{R}|}, R_{l'}^{adj})$ if $C_l \leq C_{l'}$ and for all components $t \in \{1, ..., |\mathcal{R}|\}$, the inequality $R_l^t \leq R_{l'}^t$ holds. Observe that we do not dominate on the adjacency resource. The efficiency of a labeling algorithm is a result of its ability to identify and discard subpaths that cannot contribute to an optimal path. It discards both infeasible subpaths and subpaths that may not lead to a better (feasible) solution than a given subpath. For more details on labeling algorithms and dominance principles and rules, see Irnich & Desaulniers (2005).

In Step 7, we call the $Extend(l_i, j)$ function, which extends a label l_i to a node j using predetermined extension rules. The most commonly used rules are as follows : $C_{l_j} := C_{l_i} + c_{ij}$, $R_{l_j}^t := \max\{a_i^t, R_{l_i}^t + r_{ij}^t\} \forall t \in \{1, ..., |\mathcal{R}|\}$, $R_{l_j}^{adj} := R_{l_i}^{adj} + r_{ij}^{adj}$; other rules may appear in real applications.

If the extended label l_j is feasible in terms of the SPPRC constraints and respects the current adjacency degree, it is added in Step 9 to the set of active labels in node j. Otherwise, if it violates the adjacency resource constraints while respecting the original SPPRC constraints, it is saved in node j to be used in the subsequent iteration(s). It is also saved if node j is not contained in the working graph $\bar{G}$ (mainly in the *Combination* step). We note that a label of degree k in node j is saved in $\mathcal{S}_j^k$. In Step 12, we use $\mathcal{S}_j$ instead of $\mathcal{S}_j^k$ to simplify the notation. The infeasible labels in terms of the SPPRC constraints are obviously ignored, since they are not added to $\mathcal{L}_j$ or $\mathcal{S}_j$. The motivation is to allow PAB to continue the extension of the labels that have been previously generated, instead of regenerating them from scratch at each iteration.

In Steps 13–14, we build Ω from Pareto-optimal labels of the destination node, we recompute C^{best} via $\min\{C^{best}, C_\pi : \pi \in \Omega\}$, and we return Ω.

6.4.3 Convergence analysis

We now discuss some important features of PAB. We first prove that no label is produced more than once by the algorithm. We then show that efficient labels in DP are also efficient for PAB, before concluding that the proposed algorithm terminates by finding the optimal solution, given a predetermined maximum degree of adjacency.

Proposition 15. *No label is generated more than once by PAB.*

Proof. Consider a label l of degree of adjacency k generated in iteration k. The labels saved

in the *Extension* step in iteration k are all of degree of adjacency $k+1$. Some of these labels that are saved in nodes of G^{Π} are extended in the next *Combination* step, while others are extended in the next *Extension* step associated with higher degrees of adjacency. The newly created labels are all of degree $k+t$ ($t \geq 1$). Consequently, label l will never be regenerated in future iterations. □

Proposition 16. *If a label l_i is efficient (Pareto-optimal) in node i for the DP algorithm, it is also efficient in i in one of the iterations of PAB, unless it is eliminated by cost bounding.*

Proof. Let l_i be an efficient label in node i for DP. This implies that there is no label from the source node s to node i of all degrees that dominates l_i. In particular, given a fixed degree k, there is no label of degree k that dominates l_i in node i. Label l_i is then efficient in PAB as well unless it is eliminated by cost bounding. This is true provided that all degrees of adjacency are covered by PAB. □

The next proposition shows that the objective value of the SPPRC cannot increase from one iteration to the next or within the same iteration between two calls to ILA.

Proposition 17. *For any two consecutive calls to ILA, the cost of the paths generated strictly decreases.*

Proof. At the end of each call to ILA, whether in a *Combination* step or in an *Extension* step, the best cost found C^{best} is used to tighten the cost upper bound for each node in the network. Thus, for a given node i, the cost C_l of each feasible label $l \in \mathcal{L}_i$ satisfies $C_l < U_i = C^{best} - C_{di}$. In particular, in the destination node d where $C_{dd} = 0$, we have $C_l < C^{best}$. The cost is thus strictly decreasing until optimality. □

Proposition 18. *PAB terminates by finding an optimal solution.*

Proof. Based on Proposition 16, all efficient labels in DP are efficient for a certain degree of adjacency in PAB. In particular, the efficient labels in the destination node d in DP are efficient in PAB unless they are eliminated by cost bounding, which is not the case for an optimal solution. The exactness of the algorithm is thus guaranteed once k_{max} is achieved. □

Note that as a primal method, PAB returns iteratively sets of non-decreasing primal feasible solutions leading to optimality. This feature, which is not offered by classical DP methods, has two important advantages. First, the best cost returned at the end of each iteration is used to tighten the cost bounds, which greatly reduces the combinatorial complexity. Second, the primal aspect of PAB is a good match with the requirements of the CG method. The CG

subproblems seek negative reduced cost paths, and these paths can be obtained using PAB in reasonable computational times.

6.5 Experimental results

To evaluate the efficiency of PAB, we conducted a series of computational experiments. The tests were performed on instances of the simultaneous vehicle and crew scheduling problem (VCSP) used by Haase *et al.* (2001) in their study of urban transit systems. This problem is representative of a broad class of general VCSPs both in terms of the size of the networks and the number of resources. In some airline instances, we have to deal with tens of resources, although dominance is in practice performed only on four to five. For the tests, we dominate on seven resources, a number that is large enough to simulate the most complex situations in airline transportation. Furthermore, good initial points can easily be obtained for these problems, so they provide a good framework for evaluating the efficiency of primal solution methods. We will give a brief definition of the VCSP, followed by some implementation tips, before presenting the computational results.

6.5.1 VCSP instances

6.5.1.1 VCSP overview

Given a predetermined set of bus lines in a city and a set of timetabled trips to operate on these lines, the VCSP simultaneously determines bus and crew schedules that cover the set of trips at a minimum cost. A timetabled trip is divided into several consecutive segments, called *d-trips*. At locations called *relief points* drivers can change bus lines or return home or to the depot. Empty moves called *deadheads* are used to reposition buses. The set of consecutive d-trips and deadheads performed by a driver is called a *piece of work*, and two pieces of work are separated by a *break*. A work day for a driver, called a *duty*, is a sequence of pieces of work and breaks.

Several formulations have been proposed for the VCSP. Most of them formulate the problem via a set partitioning model and solve it using branch and price. The constraints that assign exactly one driver to each d-trip form the core of the master problem. We note that the bus schedules are usually derived a posteriori from the driver schedules. To facilitate this, supplementary constraints are usually added to the master problem. In addition to these constraints, the driver schedules are restricted by a variety of rules defined by the collective agreements and internal regulations. These rules are modeled in the subproblems using resource constraints. Each subproblem is an instance of the SPPRC on a space-time network,

and it is solved to generate feasible driver schedules. A schedule is said to be feasible if it satisfies seven resource constraints : the minimum and maximum number of pieces of work ; the maximum time of a duty, of a break, and of work in a duty ; and the minimum and maximum length of a piece of work. For more details about the VCSP, see Haase *et al.* (2001).

The complexity of the SPPRC is a result of the number of resources and the length of the resource intervals. Table 6.1 is reproduced from Haase *et al.* (2001) ; it gives the lower and upper bounds of the resource intervals as specified by the work rules. These intervals and the reduced cost interval are wide, which makes the problem difficult : we may have thousands of nondominated labels per node.

Table 6.1 Work rules for a driver schedule

	Minimum	Maximum
No. of pieces	1	3 or 4
Piece length (mins)	15	300
Duty length (mins)	45	600
Work time (mins)	30	480
Break time (mins)	15	90

6.5.1.2 Instance generation and initial point

The VCSP instances differ in three parameters : the number of trips to be covered (120, 160, 200, or 240) ; the number of relief points per trip (5, 7, or 9) ; and the number of pieces of work per duty (3 or 4). We created a total of 24 instance classes : 12 with three pieces of work and 12 with four pieces. For each class, we randomly generated five instances using the generator of Haase *et al.* (2001). This gives a total of 120 VCSP instances with up to one million arcs (see Table 6.2).

The VCSP is usually solved by branch and price. We used CG to solve the LP relaxation of these VCSP instances. For each VCSP instance, we captured randomly two pricing subproblems, one from the first iterations of CG process and the second one from the last iterations. These 240 pricing subproblems form our test instances. This allows for a fair comparison between PAB and DP (same well-defined instances with well-defined reduced costs). We do not report results of the integration of PAB into the CG process to avoid side effects from the LP solver used for the master problem (mainly its impact on the dual variables used to compute the reduced costs) and other CG strategies used to accelerate convergence.

We use “3-pieces-b”, “3-pieces-e” to denote respectively instances with a maximum of three pieces of work taken from the beginning of CG and from the end. Similarly, “4-pieces-b” and

Table 6.2 List of test instances (PAB)

Instance class	# trips	# relief points	# d-trips	# arcs	# nodes
5_120	120	5	720	78989.2	50690.0
5_160	160	5	960	141286.0	91458.4
5_200	200	5	1200	220006.8	143100.4
5_240	240	5	1440	314692.6	205377.4
7_120	120	7	960	152192.4	98891.2
7_160	160	7	1280	273755.4	178959.6
7_200	200	7	1600	427856.4	280649.0
7_240	240	7	1920	612310.2	402557.6
9_120	120	9	1200	249093.6	162862.0
9_160	160	9	1600	450255.6	295783.2
9_200	200	9	2000	703604.8	463421.0
9_240	240	9	2400	1008197.8	665201.0

"4-pieces-e" denote instances with four pieces of work. We also use "b-instances" and "e-instances" to distinguish instances extracted from the beginning of CG from those extracted from the end.

We constructed an initial point by assigning to each timetabled trip a path that starts from the depot, covers the trip, and returns to the depot. These initial paths are added to $\Pi = \Pi_0$, and their arcs are added to A^{Π}. These paths may be infeasible, but they contain a good percentage of the primal information.

We choose this initial point for the following reasons. First, in crew scheduling problems, the crews do not often change vehicles, so the crew schedules have many pieces in common with the bus lines. Second, in a reoptimization context, we observe that the reoptimized paths are characterized by a slight deviation from the planned paths : a high percentage of the arcs and nodes of the initial point remain in the final reoptimized Pareto-optimal paths. Third, the construction of the initial point does not require additional computational time since all the information is available.

6.5.1.3 Theoretical results related to VCSP instances

We know from Definition 14 that a path π is k-adjacent to a set of paths Π_0 if k detours are needed to construct the path π using Π_0. Based on the initial point above, each detour corresponds to a break between two consecutive pieces of work in a schedule. The maximum number of allowed detours k_{max} is then the maximum number of daily pieces of work minus one. Hence, the adjacency resource corresponds to a resource that counts the number of pieces of work. Under this condition, we can prove that the number of labels generated by all

Extension steps in G is less than or equal to the number generated by DP. If we consider in addition the effect of cost bounding, the inequality becomes strict. Thus, PAB creates fewer labels than DP.

Proposition 19. *The number of labels created by PAB in all Extension steps is less than or equal to the number created by the DP algorithm.*

Proof. First, by Proposition 15, no label is generated more than once by PAB. Second, the feasibility conditions in terms of resource constraints are the same for the two algorithms. Third, suppose that there is a label l_i of degree k that is dominated at node i by DP but not by PAB. Let $l_i^{'}$ be the efficient label that dominates l_i. There are three cases. Case 1 : If the degree of $l_i^{'}$ is strictly greater than k, dominance is not possible, because the degree of adjacency corresponds to the number of detours, which is a counter corresponding to one of the original resources of the problem (the number of pieces of work). Case 2 : $l_i^{'}$ is of degree k. In this case, $l_i^{'}$ dominates l_i at the k^{th} iteration of PAB as well. Case 3 : The degree of $l_i^{'}$ is strictly less than k. In this case, $l_i^{'}$ is generated in the previous iterations and saved to be used to dominate l_i in the subsequent iterations. Therefore, every label eliminated by dominance in DP is similarly eliminated in PAB. Moreover, many labels can be eliminated at each iteration of PAB due to cost bounding ; this completes the proof. □

It is possible that some of the labels created in the *Combination* steps are not generated by DP. This is because the *Combination* step may generate labels of degree $k + t$ where $t \geq 1$, while only degree k is achieved by the *Extension* step. This implies that labels of degree $\geq k + t$ that may dominate those generated in the *Combination* step are not yet available for PAB. However, since the *Combination* steps are performed in very small subgraphs G^{Π}, this side effect is insignificant, as our experiments will show.

Proposition 20. *The number of calls to dominance function in all Extension steps is strictly less than the number of calls to dominance function by DP.*

Proof. Let $n = \sum_{k=0}^{k_{max}} n_k$ be the number of feasible labels in a given node of G, where n_k is the number of labels with a degree of adjacency k regarding the initial point Π_0. The number of calls to dominance function by DP is in the worst case C_n^2. This number is equal to $C_{n_k}^2$ for the *Extension* step related to an iteration k of PAB, so the total number of calls in all *Extension* steps is $\sum_{k=0}^{k_{max}} C_{n_k}^2$ which is largely smaller than C_n^2. In addition, by Proposition 19, the total number of labels generated by PAB in each node is at most equal to the number of labels generated by DP ; this completes the proof. □

Finally, the notion of a detour is not specific to the VCSP. In airline crew scheduling, we observe that pilots and copilots do not often change planes ; their rotations (pairings) thus have many pieces in common with the predetermined plane routes, and are generally similar to the schedules of previous months. If we consider these routes or previous schedules as the initial point, every deviation can be viewed as a detour. Some commercial solvers speed up the solution of these problems by limiting the generation of pairings to one detour per duty for the first 90% of the CG iterations. In these cases the number of detours in a path is two or three, corresponding to the mean number of duties in a pairing. In the final 10% of the iterations more detours are permitted, but the subproblems remain easy because there are many paths with a reduced cost close to zero, and dominance greatly reduces the number of Pareto optimal paths to consider. Even if there is no resource counting the number of pieces of work in airline crew scheduling problems, the basic notion of a detour is implicitly present.

6.5.2 Computational results

This section compares PAB with MDDPA (Himmich *et al.*, 2018a) and a standard DP algorithm (hereafter referred to as std. DP) which is the standard technique used by commercial solvers for the SPPRC. We note that MDDPA provides two search strategies, Nearest First and Best First. We use the former because it is more efficient.

All the instances have been solved on a 2.5 Ghz Intel Core i5 MacBook. The three algorithms were implemented in C++ using Boost Graph library, a well-known C++ library. For a fair comparison, we have enhanced the three algorithms with two major improvements : a topological order of the nodes that is used to explore the search space, and well tightened cost and resource upper bounds for all the nodes of the network.

6.5.2.1 Computational time

PAB performs well in terms of computational time : first, it is considerably faster than std. DP ; second, it is able to find optimal solutions earlier than MDDPA does for all the test instances ; third, it is better than MDDPA at proving optimality for b-instances while being competitive for e-instances. Tables 6.3 and 6.4 compare the computational times of the three methods. The CPU column gives the computational time in seconds, and the Opt. time column gives the time required by PAB and MDDPA to find an optimal solution before proving its optimality. As we can see, Opt. time is a fraction of CPU.

Table 6.3 Computational times for 3-piece-b and 3-piece-e instances

Instance class	b-instances					e-instances				
	std. DP	MDDPA		PAB		std. DP	MDDPA		PAB	
	CPU	CPU	Opt. time	CPU	Opt. time	CPU	CPU	Opt. time	CPU	Opt. time
5_120	3.56	2.21	1.05	1.71	1.08	1.18	0.17	0.17	0.37	0.09
5_160	9.64	5.25	2.58	3.57	2.02	3.37	1.38	0.49	1.39	0.32
5_200	21.53	10.93	6.67	8.04	6.05	6.97	4.05	1.26	3.12	0.76
5_240	33.65	21.01	16.42	14.51	6.27	12.17	3.91	2.34	3.62	0.98
7_120	8.84	4.76	3.22	3.84	2.99	2.32	0.34	0.34	0.64	0.17
7_160	18.93	15.30	10.63	9.29	5.00	8.77	2.36	0.95	2.86	0.72
7_200	47.10	30.13	25.44	20.68	12.02	18.17	6.14	3.45	5.70	2.74
7_240	95.55	56.40	34.70	31.38	12.98	26.89	8.98	6.53	7.62	3.83
9_120	15.29	9.71	5.97	6.94	6.20	5.69	1.48	0.69	1.71	0.84
9_160	41.98	24.95	14.99	14.74	10.93	16.19	7.10	4.13	6.07	2.12
9_200	95.95	55.06	37.77	35.53	20.13	42.38	12.83	4.41	12.19	6.40
9_240	176.64	114.91	61.69	56.73	21.17	56.54	18.20	17.76	12.28	7.03

Table 6.4 Computational times for 4-piece-b and 4-piece-e instances

Instance class	b-instances					e-instances				
	std. DP	MDDPA		PAB		std. DP	MDDPA		PAB	
	CPU	CPU	Opt. time	CPU	Opt. time	CPU	CPU	Opt. time	CPU	Opt. time
5_120	6.81	2.88	1.58	2.95	1.69	1.79	0.18	0.18	0.52	0.10
5_160	18.34	7.83	3.66	7.14	2.25	5.08	1.58	0.41	2.37	0.32
5_200	46.86	18.74	12.20	15.95	6.51	10.43	5.47	1.28	6.27	0.77
5_240	65.67	35.69	30.55	28.69	11.09	18.61	4.67	2.36	6.08	1.42
7_120	14.02	6.62	4.83	6.42	3.23	3.58	0.37	0.37	0.92	0.16
7_160	33.00	20.03	11.10	16.31	4.21	12.57	2.79	0.99	4.76	0.94
7_200	111.08	61.87	31.15	50.01	15.34	28.02	7.35	4.06	10.35	2.72
7_240	194.96	92.26	49.88	66.06	27.71	43.36	11.36	6.72	12.47	3.74
9_120	29.06	13.78	8.17	12.31	5.69	9.97	1.85	0.77	2.23	0.83
9_160	89.46	39.21	22.68	29.00	10.46	28.90	10.76	5.85	11.23	2.17
9_200	252.62	103.70	65.58	81.07	25.26	72.92	17.17	4.08	22.27	6.39
9_240	343.79	189.01	137.69	152.13	82.72	97.30	22.73	22.27	19.50	7.08

We compute the time reduction factors realized by PAB as the ratio of the std. DP time to the PAB time. The average ratio is about 2.53 for the b-instances and 3.21 for the e-instances. The largest reduction factors occurred for the largest instances. PAB is more efficient than MDDPA for the b-instances, since the reduction factors of the latter do not exceed 1.97 with an average of 1.89. However, MDDPA is better than PAB for the e-instances, with an average reduction factor of 4.21. Figure 6.4 shows the PAB time reduction factors for different classes of instances.

Tables 6.3 and 6.4 also give the time when an optimal solution is found by PAB and MDDPA.

PAB performs better for all the classes of instances. It returned optimal solutions in an average of 24.75% of the total std. DP time for 3-piece-b instances and about 16.42% for 4-piece-b instances. The corresponding figures for MDDPA were 39.01% and 29.74%. PAB returned optimal solutions in an average of 11.36% for 3-piece-e instances and about 7.41% for 4-piece-e instances ; the corresponding figures for MDDPA were 17.89% and 12.27%. Since the complexity of the SPPRC grows significantly with the size of the network and the number of pieces of work per duty, we conclude that the efficiency of PAB is more significant for more complex instances.

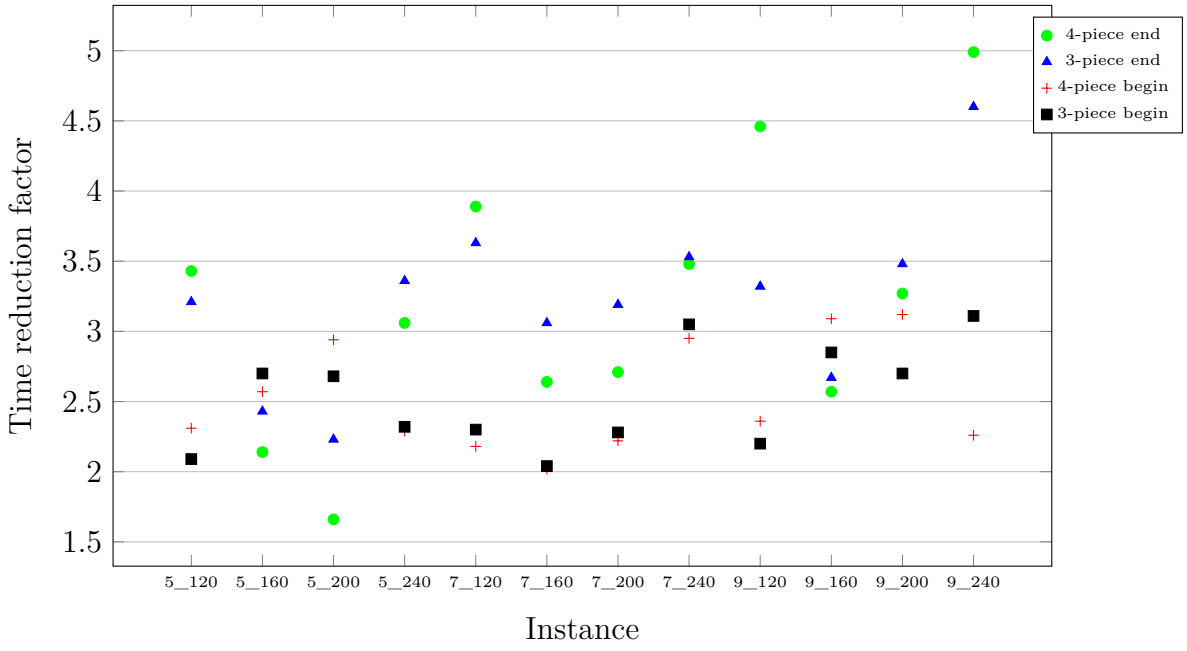

Figure 6.4 Time reduction factor (PAB vs. std. DP)

Figure 6.5 shows a typical evolution of the optimal value as a function of the computational time (expressed as a percentage of the total time consumed by std. DP). We have chosen three instances from the class 4-piece-b that are the largest in their categories : 5_240, 7_240, and 9_240. PAB is able to provide feasible paths with more than 50% of the optimal cost in less than 5% of the time required by std. DP (to return an optimal solution). Moreover, PAB needs at most 15% of this time to generate feasible paths with a cost between 80% and 95% of the optimal cost. This is observed for all the instances, regardless of the size and the number of pieces of work per duty. Also, PAB has proved its ability to return an optimal solution in a time ranging from 5% to 50% of the std. DP time. Finally, we observe that the descent of the cost is steeper for instances with more d-trips, which means that the algorithm is more efficient when we have to generate longer paths.

These results are especially significant in a CG context, where the ultimate aim is to generate feasible paths with good reduced costs as quickly as possible. Generating paths with more than 80% of the optimal solution cost in less than 15% of the total time required by std. DP will greatly reduce the CG time.

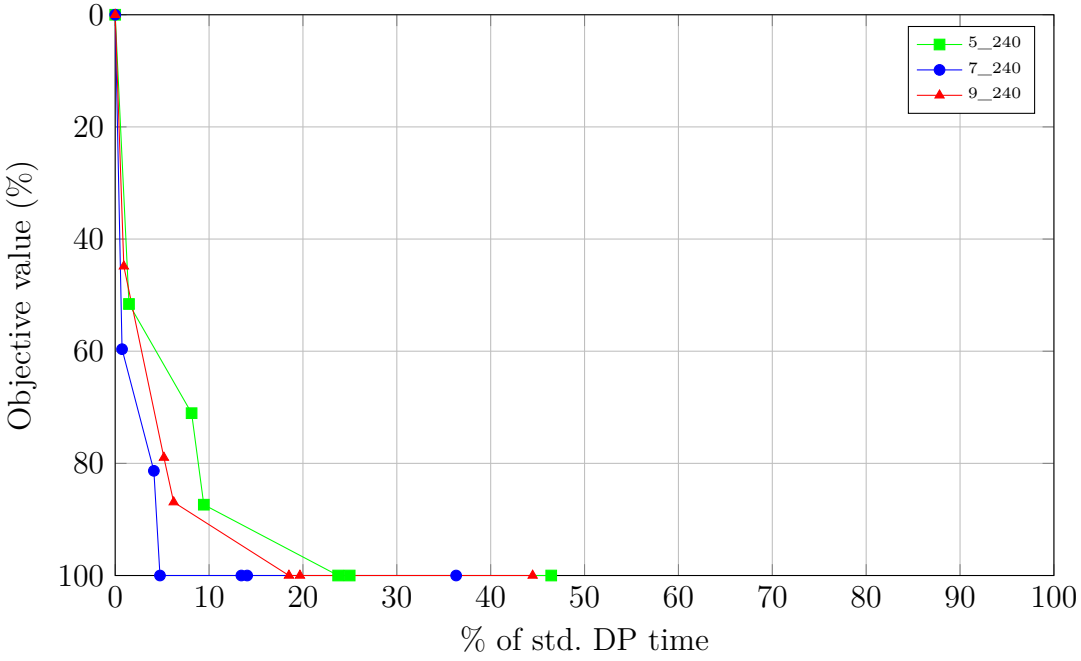

Figure 6.5 Improvement of objective value as function of % of std. DP time

6.5.2.2 Reductions in terms of labels

Tables 6.5 and 6.6 give the results related to the label performances for instances with a maximum of three or four pieces of work per duty. In these tables, "#Lab." indicates the number of created labels, and "#DCL" indicates the average number of calls to the dominance function per label. This is computed as the ratio of the total number of dominance operations to the number of created labels.

Compared to std. DP, there is a large reduction in the number of labels, which highlights the effectiveness of PAB and explains the reduction in the computational time. The label reduction factor (LRF) is between 1.47 and 2.29 for the 4-piece-b instances, and it is up to 4.02 for the 3-piece-b instances, with an average of 2.78. These ratios are significantly higher for the e-instances, giving an average overall LRF of 48.57.

Table 6.5 Label performances for 3-piece-b and 3-piece-e instances

Instance	b-instances						e-instances					
	std. DP		MDDPA		PAB		std. DP		MDDPA		PAB	
class	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL
5_120	7.2E+05	18.4	2.5E+05	13.5	3.1E+05	6.7	2.8E+05	5.3	2.4E+03	5.7	2.3E+03	2.6
5_160	1.7E+06	25.0	5.6E+05	19.9	6.0E+05	10.0	7.6E+05	9.7	1.3E+05	13.3	8.0E+04	4.6
5_200	3.3E+06	33.1	1.2E+06	34.4	1.4E+06	16.4	1.5E+06	14.4	4.6E+05	19.9	3.8E+05	9.8
5_240	5.6E+06	33.4	2.5E+06	31.0	2.4E+06	16.1	2.4E+06	16.0	2.8E+05	13.4	2.9E+05	6.4
7_120	1.6E+06	23.3	5.4E+05	15.0	6.5E+05	7.5	5.4E+05	5.0	3.2E+03	4.4	1.6E+03	3.2
7_160	3.6E+06	27.7	1.9E+06	30.0	1.4E+06	12.5	1.8E+06	13.3	2.3E+05	11.0	2.0E+05	5.0
7_200	7.3E+06	36.9	3.3E+06	41.3	3.2E+06	19.8	3.7E+06	17.2	6.1E+05	21.1	4.7E+05	4.9
7_240	1.4E+07	44.4	5.5E+06	44.6	4.3E+06	16.0	5.2E+06	17.5	6.4E+05	14.2	5.9E+05	5.4
9_120	3.0E+06	25.3	1.1E+06	20.2	1.0E+06	8.7	1.3E+06	8.8	1.0E+05	5.3	1.2E+05	3.6
9_160	7.2E+06	32.5	2.5E+06	26.5	1.8E+06	12.2	3.3E+06	13.4	6.6E+05	12.8	6.5E+05	7.3
9_200	1.4E+07	43.7	5.5E+06	36.5	5.0E+06	20.5	7.7E+06	27.8	1.2E+06	12.6	1.4E+06	9.5
9_240	2.4E+07	46.2	1.1E+07	49.7	7.8E+06	19.9	1.1E+07	20.5	1.3E+06	17.7	6.1E+05	4.9

Table 6.6 Label performances for 4-piece-b and 4-piece-e instances

Instance	b-instances						e-instances					
	std. DP		MDDPA		PAB		std. DP		MDDPA		PAB	
class	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL	#Lab.	#DCL
5_120	1.3E+06	30.34	3.6E+05	20.45	8.4E+05	8.91	4.9E+05	8.82	2.4E+03	5.98	2.8E+03	2.69
5_160	3.1E+06	42.05	9.8E+05	29.45	1.7E+06	15.16	1.3E+06	14.44	1.7E+05	14.12	3.3E+05	9.19
5_200	7.4E+06	66.39	2.1E+06	58.00	4.2E+06	25.78	2.6E+06	20.05	7.7E+05	23.36	1.4E+06	17.93
5_240	1.1E+07	62.46	4.4E+06	51.03	6.9E+06	24.40	4.2E+06	25.36	3.8E+05	17.10	8.3E+05	11.40
7_120	2.9E+06	37.11	8.3E+05	22.25	1.8E+06	10.68	9.8E+05	8.65	3.2E+03	4.40	2.5E+03	3.35
7_160	6.4E+06	42.87	2.6E+06	40.69	4.0E+06	19.68	3.1E+06	18.27	3.1E+05	12.24	7.2E+05	9.23
7_200	1.6E+07	71.98	6.1E+06	84.34	1.0E+07	41.60	6.1E+06	23.89	7.3E+05	21.88	1.5E+06	13.46
7_240	2.6E+07	75.23	9.5E+06	71.96	1.4E+07	29.60	8.9E+06	26.86	9.8E+05	20.19	1.7E+06	9.52
9_120	5.6E+06	41.26	1.7E+06	28.78	3.1E+06	13.89	2.4E+06	15.28	1.7E+05	8.70	1.9E+05	3.60
9_160	1.4E+07	55.13	4.5E+06	42.17	6.1E+06	18.93	6.3E+06	21.36	1.3E+06	18.85	2.2E+06	11.72
9_200	3.0E+07	81.24	9.6E+06	80.04	1.5E+07	35.86	1.3E+07	38.38	1.9E+06	18.72	4.4E+06	14.86
9_240	4.0E+07	69.85	1.8E+07	82.72	2.7E+07	39.03	1.8E+07	30.63	2.5E+06	18.76	1.9E+06	8.63

Figure 6.6 shows the evolution of the number of labels generated by PAB as a function of time. We have chosen the instances used for Figure 6.5 (5_240, 7_240, and 9_240). The time axis represents the time consumed by PAB as a percentage of the total std. DP solution time. The isolated marks at 100% indicate the number of labels generated by std. DP, while the linked marks indicate the cumulative number of labels generated during the PAB solution process. The largest number of labels is generated during the *Extension* step for the final iteration : it represents about 50% of the total number of labels created by PAB. Since the search spaces explored at each PAB iteration are disjoint (see Corollary 2), only the saved labels are kept in memory from one iteration to the next, while the remaining labels are safely deleted. This leads us to conclude that the memory used by PAB in a given iteration is in the worst case equal to half of the total memory required during the solution process. Consequently, we must multiply the LRF by about two to obtain the real memory reduction factor of PAB.

A direct consequence of this is the reduction in the number of calls to the dominance function. Large reduction factors have been observed for all the instances. The dominance reduction

factor varies between 2.63 and 6.68 for 4-piece-b instances and between 4.25 and 10.71 for 3-piece-b instances. These ratios are much greater for e-instances, with an average reduction factor of 115.36. This is because the number of calls to the dominance function at a given node is in the worst case a quadratic function of the number of labels at that node.

Tables 6.5 and 6.6 highlight the efficiency of PAB compared to std. DP and MDDPA in terms of the average number of dominance operations per label. PAB reduced this rate from 32.5 to 13.9 and from 56.3 to 23.6 on average for 3-piece-b and 4-piece-b instances respectively, and the corresponding values for MDDPA were 30.2 and 51.0. Similar behavior is observed for the e-instances. PAB reduced the value from 14.1 to 5.6 for 3-piece-e instances and from 21.0 to 9.6 for 4-piece-e instances, and the corresponding values for MDDPA were 12.6 and 15.4. We conclude that although MDDPA reduced the number of labels and consequently the number of calls to the dominance function, the average number of dominance calls per label was only slightly affected by this reduction. In contrast, PAB achieved a further reduction ranging between two and three in the average number of dominance operations per label. This shows that classifying labels by their degree of adjacency and prioritizing dominance between labels with the same degree is better for the solution of SPPRC. These remarkable reductions greatly reduce the computational complexity, which explains the significant time improvement achieved by PAB.

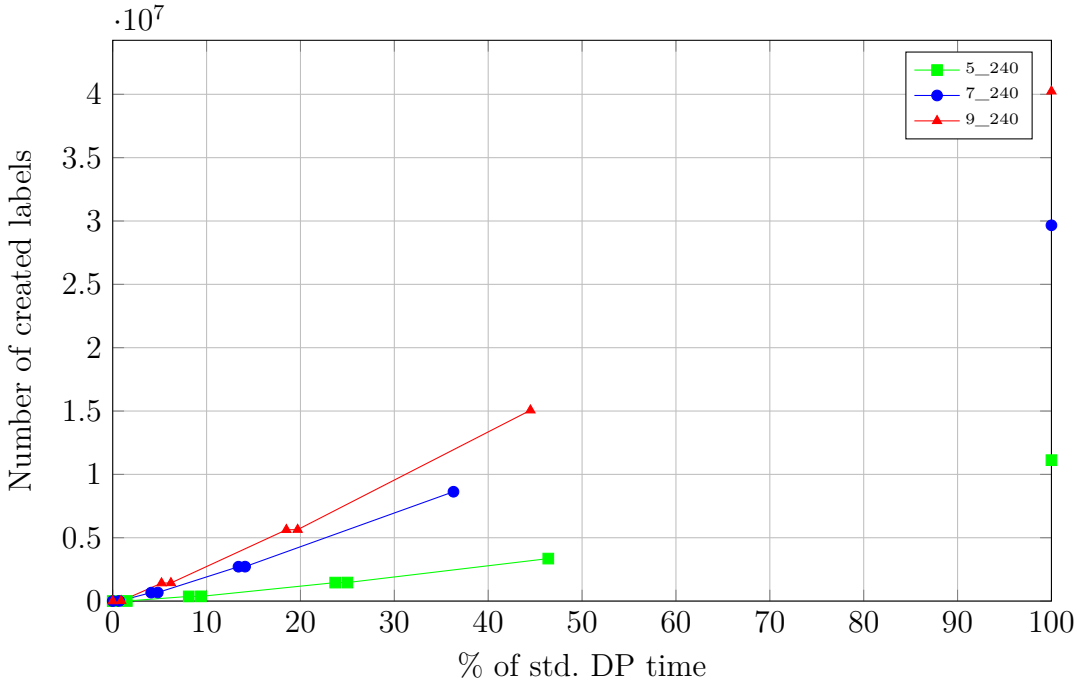

Figure 6.6 Evolution of the number of labels created by PAB

6.5.2.3 Impact of Combination step

Another interesting feature of our approach is the impact of the *Combination* step. The *Combination* step quickly improves the objective value of the subproblem between iterations, and it gives the largest part of the cost decrease. We recall that the main role of the *Combination* step, as mentioned in the last section, is to affinely combine previously generated paths to create new paths with better costs.

Table 6.7 gives the results for the PAB *Combination* step for different classes of instances. The columns "% time" give the time for the *Combination* step as a percentage of the total PAB solution time, and the columns "% gain" indicate the percentage of the decrease of the objective value that is due to *Combination* steps. To compare the performance of the *Combination* and *Extension* steps, one can extract the results for the *Extension* step from Table 6.7. These values are equal to the complementary part of the percentages for the *Combination* step, for each attribute.

Table 6.7 Details of Combination step

	b-instances				e-instances			
Instance	3-piece-b		4-piece-b		3-piece-e		4-piece-e	
class	% time	% gain	% time	% gain	% time	% gain	% time	% gain
5_120	21.74	72.69	18.73	69.73	44.41	100.00	45.27	100.00
5_160	20.77	79.47	15.53	79.69	35.42	97.14	32.14	97.14
5_200	16.43	69.31	12.34	67.41	28.67	95.97	23.59	95.97
5_240	14.13	71.01	10.64	67.26	32.05	85.87	29.86	85.87
7_120	20.61	71.75	17.54	69.56	44.58	100.00	49.22	100.00
7_160	16.14	78.32	12.48	83.23	34.81	90.86	33.59	90.86
7_200	13.91	66.63	10.66	69.86	28.52	83.49	22.89	83.49
7_240	13.42	70.36	9.25	69.19	32.31	73.24	29.97	73.24
9_120	19.40	75.49	15.44	75.49	39.45	84.72	42.94	84.72
9_160	17.87	83.03	12.41	83.60	31.21	93.54	29.01	91.24
9_200	12.63	69.59	10.36	69.46	24.83	78.32	22.79	78.32
9_240	11.49	67.91	7.38	65.76	30.27	78.90	26.45	78.90

Table 6.7 reveals that, for both b- and e-instances with either three or four pieces of work, the *Combination* step has better performance than the *Extension* step. For b-instances, the combination time ranges between 7.4% and 21.7% with an average of 14.6% of the total PAB solution time. This percentage decreases slightly with the number of pieces, for all classes of instances. In contrast, the extension time often represents more than 78.3% of the PAB time. For e-instances, the total time for the *Combination* step is about 33.1% on average for all classes of instances. The largest improvement in the objective value (between 65.7% and 83.6% for b-instances and 73.2% and 100% for e-instances) was obtained by the *Combination* step.

In summary, for b-instances, on average 72% of the optimal value requires only 15% of the time, while 85% of the time is spent finding the remaining 28% of the optimal value. For e-instances, the *Combination* step requires on average 33% of the PAB time to return 88% of the optimal value, while the remaining 12% is obtained by the *Extension* step in 67% of the time.

An explanation of these results is that, unlike the *Extension* step, the *Combination* step is performed in relatively small subnetworks that are full of primal information. The numbers of labels and dominance operations for the *Combination* steps are low compared to their values in the *Extension* steps. Since the main contribution of this step is to affinely combine existing paths to produce new ones (Proposition 14), we conclude that the computational results clearly support our theoretical assertions and justify the use of affine combinations.

Since all the paths generated by the *Combination* step are affine combinations of previously generated paths, it is clear that no improvement could occur via the Combination technique if there were no good paths previously found by the Extension technique. PAB is consequently an intelligent combination of two techniques (*Combination* and *Extension*) that complement each other, resulting in a primal method that efficiently solves the SPPRC.

6.6 Conclusion

In this paper, we have considered the SPPRC. The proposed PAB algorithm is a new approach based on an iterative exploration of the search space. Our polyhedral study allowed us to take advantage of some properties of the problem. In particular, we use the notion of adjacency to restrict the search process to a limited space. In addition, using the concept of affine combinations, we have shown that better paths can be easily generated by combining existing paths.

We evaluated our method on VCSPs taken from the literature. A comparison with the standard DP method has indicated the performance of our approach. PAB reduces the solution time for all the test instances, with the reduction factor varying between two and five on average. As a primal method, PAB converges to optimal solutions faster than MDDPA does, and proves their optimality earlier than MDDPA for b-instances while being competitive for e-instances. Moreover, PAB has shown its ability to generate good paths with more than 50% of the optimal cost in less than 5% of the total time required by the standard DP approach. This result shows that the PAB algorithm is appropriate for solving the subproblems using a CG method since the aim of the CG subproblems is to find good paths as quickly as possible.

This primal paradigm opens up new interesting research tracks for i) efficiently solving more

general SPPRCs with nonlinear extension function, covering thus a wider range of CG applications, and ii) reducing "the curse of dimensionality" encountered in general when solving with DP.

Acknowledgements

This work was supported by a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Kronos Inc. The authors would like to thank these organizations for their support and confidence.

CHAPITRE 7 ARTICLE 3 : PRIMAL COLUMN GENERATION FRAMEWORK FOR VEHICLE AND CREW SCHEDULING PROBLEMS

Le texte de ce chapitre est celui de l'article (Himmich *et al.*, 2018c) :

Primal Column Generation Framework for Vehicle and Crew Scheduling Problems

Ilyas Himmich, Issmail El Hallaoui, François Soumis
GERAD et École Polytechnique de Montréal
Département de Mathématiques et Génie Industriel
C.P. 6079, Succ. Centre-Ville
Montreal, Quebec, Canada H3C 3A7
ilyas.himmich, issmail.elhallaoui, françois.soumis@gerad.ca

publié dans les cahiers du GERAD et soumis à

Networks

date de soumission : Septembre 2018

Abstract

The primal adjacency-based algorithm and the multi-directional dynamic programming algorithm are two exact methods that have recently been developed to efficiently solve the shortest path problem with resource constraints. These methods are primal in the sense that they are able to produce sequences of feasible solutions using iterative exploration of the search space. Since the shortest path problem with resource constraints often appears as a subproblem in the solution of vehicle and crew scheduling problems using column generation, we propose a new Primal Column Generation framework that embeds these primal methods in a column generation scheme. The Primal Column Generation yields at each iteration a good cost improvement in a very limited time by solving an appropriate restricted subproblem. The use of a primal approach introduces a self-acting ability and a large degree of flexibility. Computational experiments on vehicle and crew scheduling problem instances show that the proposed Primal Column Generation is able to find optimal solutions while reducing the time spent solving the subproblems by factors of up to seven compared to the standard column generation algorithm. This leads to significant improvements in the overall solution times, with an average reduction factor of 3.5.

Keywords : Column generation, Subproblems, Shortest path problem with resource constraints, Dynamic programming, Primal paradigm.

7.1 Introduction

One of the most important problems arising in mass transit systems is the scheduling of the available resources in order to cover at a minimum cost a set of predetermined services or trips. Indeed, vehicles and crews are the two main resources considered by transport companies ; the way in which they are used has a strong impact on the cost and quality of the service. Vehicle scheduling problems find a least-cost set of vehicle routes that cover all the required trips and obey specific feasibility rules on, e.g., the vehicle capacity, trip ordering and maintenance requirements. Similarly, crew scheduling problems assign crew members to the vehicle itineraries resulting from the vehicle scheduling, with the goal of minimizing cost while respecting the requirements imposed by the collective agreements and internal safety regulations.

Vehicle and crew scheduling problems (VCSPs) are complex because of their size and the nature of the various feasibility rules. They give rise to very large mixed integer programming

problems that are difficult to solve. The most popular method for these problems is column generation (CG) embedded in a branch and bound search tree. CG takes advantage of the structure of the problem and divides it into a coordinator problem, called the master problem (MP), and one or more column generators, called subproblems (SPs). Linking constraints, such as task-covering constraints, are considered in the MP, while the constraints that are separable by vehicle or crew member are treated in the SPs.

The MP usually corresponds to a set covering or set partitioning problem with additional constraints. It is traditionally solved using linear programming methods, and each SP is a shortest path problem with resource constraints (SPPRC). The SPPRCs are usually solved using dynamic programming (DP). Unfortunately, DP becomes time-consuming and inefficient on real-world instances with networks of hundreds of thousands of arcs and dozens of resource constraints.

Several heuristics have been developed to handle this problem of dimensionality. Most of these techniques select subsets of the labels or dominate on a subset of the resources. Since the state removal is not based on mathematical deductions, the solution obtained by CG for the linear relaxation may be at an unknown distance of optimality. Many solvers implement these heuristics in the first iterations of the CG scheme, when it is easy to find columns with negative reduced costs. Since the number of iterations is not known in advance, it is necessary to monitor the solution process when applying these heuristics. Thus, most programmers tune their solvers with several parameters.

Clearly, it suffices for the SPs to feed the MP with columns with sufficiently negative reduced costs. However, we must ensure global optimality at the end of the solution process. Thus, the goal is to use approximate methods to reduce the computational time without affecting the quality of the solution.

The efficiency of the CG process depends on the method used to solve the SPs. We therefore propose to incorporate two exact alternatives to classical DP algorithms, namely, the primal adjacency based (PAB) algorithm and the multi-directional DP algorithm (MDDPA). These methods (Himmich *et al.*, 2018a,b) are based on the following idea : first split the state space into disjoint subspaces ; then explore the subspaces iteratively in such a way that each iteration learns from the results of the previous ones. These methods have been tested on SPPRC instances extracted form different iterations while solving the VCSP using CG, and promising results have been obtained.

The two methods are primal in the sense that they are able to return feasible solutions at different stages while solving the SPs for a given CG iteration. Thus, it is possible to terminate the solution process whenever the current columns are judged to be good enough, without

proving optimality. We believe that this approach can dramatically reduce the number of generated states, which accelerates the solution process of the SPs. The premature termination may also save considerable SP computational time.

The local efficiency of these primal methods could be altered in a CG context by side effects related to the quality of the dual solution provided by the MP. The goal of this paper is to demonstrate the global efficiency of these methods within CG. The main contributions are as follows :

1. We propose a general framework called primal column generation (PCG) as an alternative to the standard CG framework. PCG does not rely on parameter settings or human intervention. Moreover, it gives the SP solver the ability to determine when there is no need to continue.
2. The PCG framework is the first implementation of our methods within a general CG scheme. We also propose adaptations related to the initial point and the optimality criteria.
3. We perform tests on instances of the VCSP given by Haase *et al.* (2001). We solve the linear relaxation using our methods within a PCG framework, and we compare the results with those obtained by standard CG using DP to solve the SPs. The results show the effectiveness of PCG : it reduced both the SP time and the overall solution time.
4. We answer two open questions : Is the local efficiency of our methods the same for all CG iterations ? How do side effects influence the performance of our methods within a PCG framework ?

The paper is organized as follows. Section 7.2 presents a detailed definition of the VCSP and a literature review. Section 7.3 describes the classical CG algorithm for this problem. In Section 7.4, we introduce our PCG framework. Experimental results are given in Section 7.5, and Section 7.6 provides concluding remarks.

7.2 Vehicle and crew scheduling problem

This section presents a definition of the problem, a classification of the different methods used in the literature to solve it, and introduces its mathematical formulation.

7.2.1 Problem definition

The VCSP finds a minimum-cost set of vehicle routes and crew schedules that covers a predetermined set of services or trips within a fixed planning horizon. We distinguish two problem classes : the single-depot and the multi-depot VCSP. In the multi-depot version, each depot corresponds to a subset of vehicles and crew members. Different transportation companies may have various additional requirements, so it is difficult to provide a general problem definition.

Without loss of generality, we assume that the set of bus lines is predetermined and each line must be served many times over a one-day horizon to meet the passenger demand. Therefore, with each bus line we associate a set of timetabled trips. Each trip must be covered by exactly one vehicle, and a bus can move without passengers from the end of one trip to the beginning of the next. These empty moves are called *deadheads*. Hence, a vehicle schedule or route is composed of a sequence of trips and deadheads, starting and ending at the same depot.

Each trip is divided into *d-trips* or *tasks*, delimited by stops called *relief points*. A relief point is a location where driver changes are permitted. The set of tasks performed by a driver on the same bus is called a *piece of work*. Drivers must take breaks at relief points between two consecutive pieces of work or at the depot. Therefore, the driver's working day, called a *duty*, is a succession of d-trips and breaks.

The set of all driver duties forms the crew schedule. In a feasible crew schedule each d-trip is assigned to a duty, each duty starts and ends at the same depot, and each duty is performed by exactly one crew member. The duty must also satisfy rules concerning the maximum working time, the number of breaks, the minimum break duration, the total work duration, and the number of pieces of work.

The total cost is the sum of the cost of the vehicle routes, the cost of the duties, and various penalties and fixed costs. The cost of the routes includes the cost of fuel, amortization, and bus repairs. The duty cost is composed primarily of driver salaries and overtime payments.

7.2.2 Literature review

Even the single-depot VCSP problem is NP-hard (Fischetti *et al.*, 1989). It is traditionally solved sequentially ; the vehicle scheduling problem is first solved to generate vehicle routes, and then the crew scheduling problem is solved to assign the routes to drivers. This sequential procedure was strongly criticized by Ball *et al.* (1983). They recommended prioritizing the construction of crew scheduling because the problem is more difficult. An integrated approach, in which the vehicles and crews are simultaneously scheduled, was proposed by Freling *et al.*

(1995). Integrated approaches are more efficient than the sequential method, and there is a growing need to synchronize vehicles and crews in practice.

In this paper, we focus on the simultaneous VCSP with a single depot and a homogenous fleet of vehicles. Henceforth, we use the acronym VCSP to refer to this specific version. It is NP-hard and meets the needs of a wide range of transportation companies in small to medium mass-transit systems.

Several VCSP models and solution methods have been proposed in the literature. Most are based on heuristics that can be classified into three categories (see Freling *et al.*, 2003) : 1. Scheduling the vehicles during a heuristic approach to crew scheduling ; 2. Taking crew considerations into account during the vehicle scheduling process and subsequently deriving the crew schedules ; 3. Completely integrating the vehicle and crew scheduling.

Methods in the first category are the most popular. They are mostly based on the heuristic procedure proposed by Ball *et al.* (1983). This heuristic decomposes the problem into three steps, each corresponding to a matching problem. The first step generates a set of pieces of work with a duration less than an upper bound T. The second step combines pairs of these pieces into partial duties, and the third step groups the partial duties to generate feasible complete duties. Finally, the vehicle schedules are deduced by omitting the crew-only arcs. A similar three-phase strategy is presented by Patrikalakis & Xerocostas (1992). In the first phase, a set of partial crew duties covering the timetabled trips is generated by solving a set covering problem. Based on these partial duties, a minimum cost flow problem is solved in the second phase to determine a set of vehicle schedules. Complete duties are generated in the third phase by reconsidering the available partial duties.

The first contribution in the second category was that of Scott (1985). He derived crew schedules from vehicle schedules by making minor changes to the latter based on the crew costs. The results show a slight decrease in the estimated crew costs. For a detailed review of approaches in the first two categories, see Freling *et al.* (2003).

Full integration of vehicle and crew scheduling was introduced by Freling *et al.* (1995). They proposed a new integer linear formulation with three sets of constraints : set partitioning constraints for the crew schedules, quasi-assignment constraints for the vehicle schedules, and linking constraints to ensure compatibility between the vehicle and crew schedules. They developed two algorithms : the first uses CG to dynamically generate the crew schedules, and the second generates all possible crew schedules at the start of the process. Haase & Friberg (1999) proposed the first exact solution method for the integrated VCSP. This approach is based on a set partitioning formulation with additional linking constraints. Although the authors used a sophisticated methodology, embedding CG, cuts, and clique generators in a

branch-and-bound scheme, only small instances (up to 20 trips) were solved to optimality. Two years later, Haase *et al.* (2001) presented a new set partitioning formulation for the crew schedules, incorporating side constraints for the bus schedules. They constructed the crew schedules using CG in a branch-and-bound scheme and then derived the vehicle schedules in polynomial time. Instances with up to 150 trips were solved to optimality using this approach. More details about the formulations and solution methods of the third category is given by Grötschel *et al.* (2003). For a detailed review on VCSP with one or several depots, the interested reader is referred to Bunte & Kliewer (2009).

7.2.3 Mathematical formulation

The model for an integrated VCSP depends on the requirements of the specific application. In this section, we present the formulation proposed by Haase *et al.* (2001). This formulation corresponds to a set partitioning problem with additional constraints. It constructs crew schedules while taking into account vehicle considerations, so that the vehicle schedules can subsequently be derived in polynomial time.

Before presenting the formulation, we introduce some notation. Let T be the set of trips and D the set of d-trips. Let H be the set of times $h \in H$ at which a bus must leave the depot to travel to the start location of a trip and arrive exactly at the trip start time. We assume that there are several duty types. We denote by Ω_k the set of all feasible solutions (paths) p representing duties of type k, where K is the set of duty types. Furthermore, we associate with each path p four binary parameters : a_p^d takes the value 1 if path p covers d-trip d and 0 otherwise ; s_p^t takes the value 1 if path p contains travel to the start location of trip t and 0 otherwise ; e_p^t takes the value 1 if path p contains travel from the end location of trip t and 0 otherwise ; and q_p^h takes the value 1 if path p contains travel (or waiting) starting at or before time $h \in H$ and ending after h and 0 otherwise. Each vehicle has a fixed cost of use c, and each duty for path p has operating cost c_p. The binary variables θ_p^k indicate whether or not the duty of type k represented by path p is assigned to a driver, and the integer variable N counts the number of buses used to cover the timetabled trips.

The formulation is as follows :

$$\textit{Minimize} \qquad cN + \sum_{k \in K} \sum_{p \in \Omega_k} c_p \theta_p^k \tag{7.1}$$

s.t.

$$\sum_{k \in K} \sum_{p \in \Omega_k} a_p^d \theta_p^k = 1 \quad \forall d \in D \tag{7.2}$$

$$\sum_{k \in K} \sum_{p \in \Omega_k} s_p^t \theta_p^k = 1 \quad \forall t \in T \tag{7.3}$$

$$\sum_{k \in K} \sum_{p \in \Omega_k} e_p^t \theta_p^k = 1 \quad \forall t \in T \tag{7.4}$$

$$\sum_{k \in K} \sum_{p \in \Omega_k} q_p^h \theta_p^k \leq N \quad \forall h \in H \tag{7.5}$$

$$\theta_p^k \in \{0, 1\} \qquad \forall k \in K, \forall p \in \Omega_k \tag{7.6}$$

The objective function minimizes the total cost, which is the sum of the costs of the vehicles and the wages of the crews. Constraints (7.2) ensure that each d-trip is covered exactly once. Constraints (7.3) and (7.4) guarantee that exactly one vehicle arrives at the start location and leaves the end location of each trip, respectively. Constraints (7.3) and (7.4) are flow conservation constraints for the fleet of vehicles, because only d-trip arcs represent vehicle movements between relief nodes. Finally, constraints (7.5) compute the number of vehicles in use at each departure time. Since the fixed cost of a vehicle is nonnegative and the variables θ_p^k, $\forall k \in K$, $\forall p \in \Omega_k$, are binaries, the optimal solution gives the number of vehicles N allowing the completion of all the timetabled trips at a minimal cost while satisfying the crew coverage of the d-trips.

7.3 Standard column generation

The formulation above assumes that all the feasible duties in $\Omega = \cup_{k \in K} \Omega_k$ are known in advance. However, as the number of timetabled trips grows, the size of the corresponding network increases rapidly, which leads to a large number of feasible duties. It quickly becomes intractable to explicitly generate all the duties, and the resulting model would have a huge number of variables and be difficult to solve. The most popular alternative is CG embedded in a branch-and-bound scheme.

7.3.1 Overview of CG

CG is based on Dantzig–Wolfe decomposition (Dantzig & Wolfe, 1960). It splits the corresponding problem into a MP and one or more SPs, one for each duty type. For VCSPs, the MP is simply the linear relaxation of (7.1)–(7.6). It considers only the global linking constraints, while the SPs handle the local constraints related to the feasibility of the vehicle and crew schedules.

CG solves at each iteration a reduced version of the MP, called the restricted master problem (RMP), which involves only a small subset of variables $\Omega^{'} \in \Omega$. The RMP is usually solved using linear programming, which provides, at each iteration, a pair of primal and dual solutions. CG uses the dual variable values to update the reduced costs of the arcs in the SP networks. Formally, let $G^k(V^k, A^k)$ be the network corresponding to duty type k. If $\alpha = \{\alpha^d | d \in D\}$, $\beta = \{\beta^t | t \in T\}$, $\gamma = \{\gamma^t | t \in T\}$, and $\delta = \{\delta^h | h \in H\}$ are the vectors of the dual variables associated with the constraints (7.2)–(7.5) respectively, and c_{ij} is the cost of an arc $(i, j) \in A^k, k \in K$, the reduced cost of this arc is computed as follows :

$$\bar{c}_{ij} = c_{ij} - \sum_{d \in D} a^d_{ij}\alpha^d - \sum_{t \in T} s^t_{ij}\beta^t - \sum_{t \in T} e^t_{ij}\gamma^t - \sum_{h \in H} q^h_{ij}\delta^h.$$

We note that the constants a^d_{ij}, s^t_{ij}, e^t_{ij}, and q^h_{ij} are defined as in Formulation (7.1)–(7.6), except that we replace the path index p by the index ij of an arc (i, j).

The reduced cost of a given path is the sum of the reduced costs of the included arcs. The role of the SPs is to generate new feasible paths with negative reduced costs. These paths are added as columns in the RMP, i.e., the subset $\Omega^{'}$ is augmented, and a new iteration is launched. The algorithm stops when no negative-reduced-cost path can be found, which is consistent with the simplex optimality criterion.

7.3.2 Standard DP algorithm for SPs

For VCSPs, the SPs are usually instances of the SPPRC. Consider a connected acyclic network $G(V, A)$, where V is the set of nodes, including the source node s and the destination node d, and A is the set of arcs. We assume that the set of nodes V is topologically ordered, and each node is indexed by its rank in this order. In particular, 1 and $|V|$ denote s and d respectively. Let $\mathcal{R}$ be the set of resource constraints. In addition to the cost c_{ij}, each arc $(i, j) \in A$ has an $|\mathcal{R}|$-dimensional resource consumption vector $(r^1_{ij}, r^2_{ij}, ..., r^{|\mathcal{R}|}_{ij})$. Similarly, we associate with each node $i \in V$ a resource interval for each resource $t \in \mathcal{R}$. The SPPRC finds a least cost path among all the paths from s to d that satisfy the resource constraints induced by $\mathcal{R}$.

The standard approach for the SPPRC is DP. The basic DP algorithm was devised by Desrochers & Soumis (1988a) as an extension of the well-known Bellman–Ford algorithm. It explores the search space by assigning states to each node. A state in node i corresponds to a subpath from the source node to node i. Each state is represented by a multidimensional vector $l = [C_i, R^1_i, R^2_i, ..., R^{|\mathcal{R}|}_i]$, called a *label*, where C_i and $R^t_i, t \in \mathcal{R}$ are the total cost and the resource consumption of each resource $t \in \mathcal{R}$ over all the arcs on the corresponding partial path from s to i.

Algorithm 7.1 gives the pseudocode for the standard $DP(G, \mathcal{L}, \Pi)$ procedure, where $\{\mathcal{L}_i, i \in V\}$ is the set of labels initialized with a trivial label $l_1 = [0, 0, ..., 0]$ at the source node, and empty sets at the other nodes, and Π is the set of feasible paths generated by the procedure.

Algorithm 7.1: Dynamic Programming algorithm $DP(G, \mathcal{L}, \Pi)$

```
for all i ∈ V do
    Dominance(L_i)
    for all j ∈ V do
        L_j ← Extension(L_i, j)
if L_d ≠ ∅ then
    Build Π from L_d
```

The $DP(G, \mathcal{L}, \Pi)$ procedure explores the state space by calling $Extension(\mathcal{L}_i, j)$ for each node $i \in V$. This function creates new labels by extending the existing ones at node i to its successor nodes $\{j \in V | (i, j) \in A\}$, checks the feasibility of the new labels, and discards infeasible options. A label is feasible if it corresponds to a subpath that respects the resource constraints. In addition to the feasibility restrictions, a state may be fathomed if it cannot lead to an optimal solution. These decisions are made using $Dominance(\mathcal{L}_i)$. Several dominance rules may be considered depending on the requirements of the problem. The most common dominance rule is given in Definition 15.

Definition 15. *Let l_1 and l_2 be two feasible labels associated with two partial paths from s to node i. We say that l_2 is dominated by l_1 if and only if $C_1 \leq C_2$ and $R_1^t \leq R_2^t$ $\forall t \in \mathcal{R}$ and at least one inequality is strict.*

DP methods are exact, able to generate feasible paths, and efficient for small and medium instances. However, their performance relies heavily on the effectiveness of the fathoming techniques used to reduce the size of the state space. The number of states increases rapidly with the size of the problem and in the worst case grows exponentially with the number of resources. This gives rise to large spaces, which are computationally expensive to explore. We present below an alternative approach.

7.4 Primal column generation framework

This section provides the preliminaries needed to explain our PCG framework for the VCSP. We first outline the framework and then introduce the primal methods we implement within PCG, namely PAB, MDDPA using a *Nearest First* (NF) strategy and MDDPA using a *Best First* (BF) strategy. For simplification, we assume in this section that there is a single duty type and hence only one SP.

7.4.1 General overview

The PCG framework has three components : an RMP, an improved decomposable version of the SP, and a control component to manage the dependencies between them. The three-component structure is illustrated in Figure 7.1.

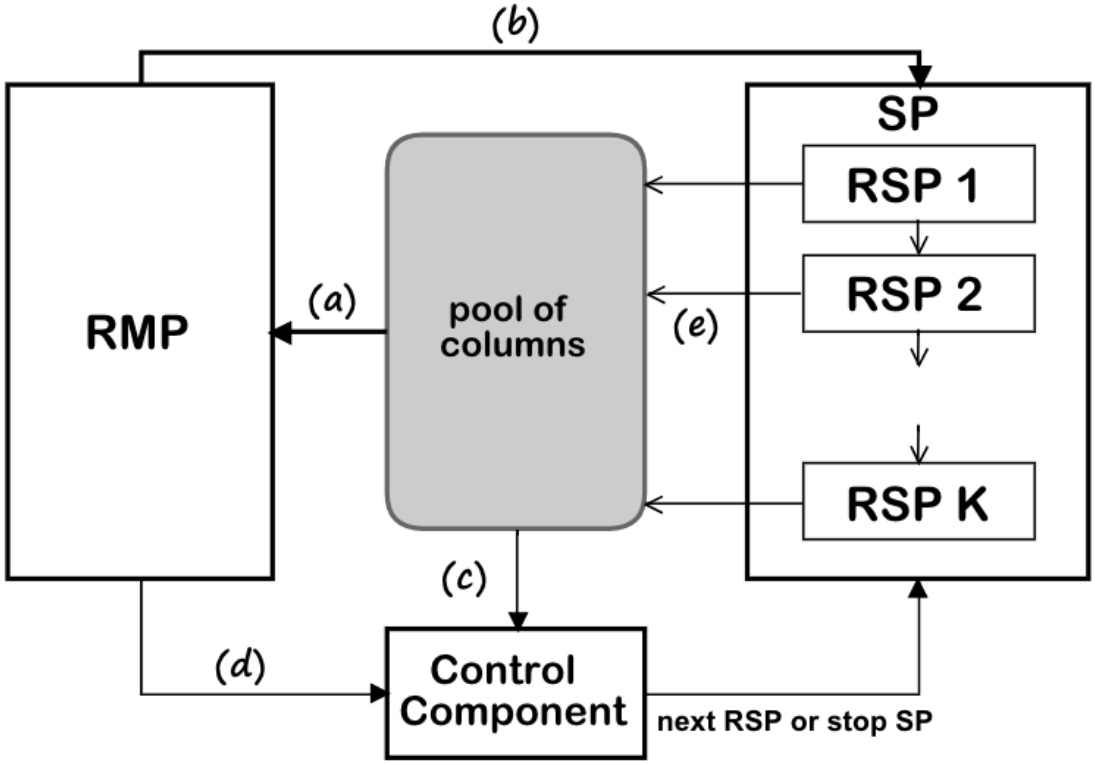

Figure 7.1 PCG framework

The RMP is similar to that described in the previous section. It is initialized with a small number of columns Ω^0 giving an initial feasible solution. It manages two flows at each iteration : the set of negative-reduced-cost columns that are sent from the SP to the RMP (flow (a)), and the current dual solution used to update the reduced costs on the arcs in the SP networks (flow(b)).

The major enhancements we propose for PCG are related to the solution of the SPs. The framework is based on a decomposition of the SP state space into several disjoint subspaces. Each subspace leads to a restricted subproblem (RSP) that is solved separately. The SP becomes a set of small RSPs, and each subproblem is potentially able to generate feasible paths. These are subsequently transferred to the RMP.

Two decompositions are proposed for the SP state space. The first is based on a measure of

distance derived from the *degree of adjacency* of a given path in relation to an initial point, and the second uses the distance of the subpaths from the sink node. These decompositions are explained in Sections 7.4.2.1 and 7.4.3.1 respectively.

The aim of solving the SP in a CG context is to provide the RMP with sufficiently negative reduced cost feasible paths. The PCG solver does not need to solve all the RSPs in a given CG iteration ; it can stop the solution process after each RSP, if a set of good feasible paths has been generated (flow (e)). The quality of the paths is evaluated using the control component of the framework.

The control component determines whether or not the current set of paths should be transferred to the RMP. If the set is not yet adequate, the algorithm continues to the next RSP. To make the decision, a control component procedure $CCP(\delta^r, \Omega^r, C^{best})$ is called after each solving of a new RSP. This procedure uses information about the number and the quality of the paths (flow (c)). This information is mainly extracted from the pool of negative-reduced-cost columns Ω^r in iteration r of CG. The most negative reduced cost in Ω^r is denoted C^{best}. The CCP procedure also uses information from the RMP (flow (d)), namely the improvement of the objective function δ^r between the last two CG iterations. Formally, $\delta^r = z_{r-1} - z_r$ where z_{r-1} and z_r are the objective values in iterations $(r-1)$ and r respectively. The CCP requires two parameters $\bar{\delta}$ and $\bar{C}$, the former is a lower bound of the objective function improvement, while the latter is an upper bound of the best reduced cost. These parameters are adjusted once in the beginning of the solution process.

The CCP is defined in Algorithm 7.2. It starts by initializing a boolean variable *stop* by a *true* value. If the improvement of the objective value is not sufficient, the CCP decides to continue solving the next RSP by affecting a $false$ value to *stop*. Otherwise, it evaluates the quality of the available negative reduced cost columns and takes a decision according to it.

Algorithm 7.2: Control Component Procedure $CCP(\delta^r, \Omega^r, C^{best})$

```
stop ← true
if δ^r < δ̄ then
    stop ← false
else
    if Ω^r ≠ ∅ and C^best < C̄ then
        stop ← true
    else
        stop ← false
return stop
```

Finally, we emphasize that apart the two parameters used by the CCP, the PCG does not

require any additional parameter. This makes the proposed CG framework less dependent on human intervention compared to classical CG algorithms that need numerous parameters and many static and dynamic adjustments.

7.4.2 Multi-directional DP algorithm

We now discuss the state space decomposition and then introduce MDDPA.

7.4.2.1 MDDPA decomposition

We use the notation of Section 7.3.2, and we assume here again that the set of nodes V is topologically ordered, and each node is indexed by its rank in this order. We first build sets of labels $\mathcal{S} = \cup_{i \in V} \mathcal{S}_i$, where each label in $\mathcal{S}_i$ represents a feasible subpath from s to i. These sets are constructed in an initialization step : we first define a restricted subgraph $\bar{G}(\bar{V}, \bar{A})$ of $G(V, A)$ such that : $\bar{V} \subset V$, $\bar{A} \subset A$, $s \in \bar{V}$, $d \notin \bar{V}$ and for each arc $(i, j) \in \bar{A}$, we have $i \in \bar{V}$ and $j \in \bar{V}$. Then we call an improved version of DP called the Label Storing Procedure $LSP(\bar{G}, \mathcal{L}, \mathcal{S}, \Pi)$, where $\mathcal{L} = \cup_{i \in V} \mathcal{L}_i$ is the set of active labels initialized with a trivial label ($l_1 = [0, 0, ..., 0]$ at the source node and empty sets at the other nodes) and Π is the set of feasible paths generated by the procedure. The LSP procedure is performed once at the beginning of the MDDPA. It calls $Dominance(\mathcal{L}_i)$ and $Extension(\mathcal{L}_i, j)$ for each node $i \in \bar{V}$ and fills the sets $\mathcal{S}_i$, $i \in V$. Formally, let $A^+(\bar{G}) = \{(i, j) \in A \setminus \bar{A} | i \in \bar{V}\}$ be the set of arcs leaving $\bar{G}$. The newly created labels are stored in a temporary list $\mathcal{T}_j$. If a label is extended using an arc $(i, j) \in A^+(\bar{G})$, it is stored in $\mathcal{S}_j$. Otherwise, it is added to the set of active labels that may lead to feasible paths in $\bar{G}$. The main steps of LSP are summarized in Algorithm 7.3.

Algorithm 7.3: Label Storing Procedure $LSP(\bar{G}, \mathcal{L}, \mathcal{S}, \Pi)$

```
S ← ∅
for all i ∈ V̄ do
    Dominance(L_i)
    for all j ∈ V do
        T_j ← Extension(L_i, j)
        if (i, j) ∈ A⁺(Ḡ) then
            S_j ← S_j ∪ T_j
        else
            L_j ← L_j ∪ T_j
if L_d ≠ ∅ then
    Build Π from L_d
Return S_i, ∀i ∈ V
```

We note that $\bar{G}$ may be constructed in different ways. In this paper, we construct $\bar{G}$ using the set of nodes and arcs on the paths for the nondegenerate variables (columns) of the current basic solution of the RMP. In particular, at the first iteration where there is no basic solution, $\bar{G}$ can be constructed by assigning to each trip an artificial path that starts from the depot (source node), covers the trip, and returns to the depot (destination node). Of course, we don't consider in $\bar{G}$ the destination node and the arcs entering to it in order to fulfill the aforementioned conditions.

The second step of the state space decomposition process uses a set of cocycle constraints, defined as follows :

Definition 16. *Consider a directed acyclic network $G(V, A)$. The cocycle constraint of order $k \in \{1, 2, ..., |V| - 1\}$ is the constraint $\sum_{(i,j)\in Co_k} x_{ij} = 1$, where $Co_k = \{(i, j) \in A | i \leq k < j\}$ is the k^{th} cocycle.*

The cocycle constraints for $k \in \{1, 2, ..., |V| - 1\}$ are equivalent to the flow conservation constraints (Himmich *et al.*, 2018a). This means that covering each cocycle exactly once suffices to ensure the connectivity of a given path from s to d.

We denote by i_0 the index of the first node whose set of stored labels is nonempty. Formally, $i_0 = \text{argmin}_{\mathcal{S}_i \neq \emptyset} \mathcal{S}_i$. Moreover, each label in $\mathcal{S} = \cup_{i\in V} \mathcal{S}_i$ is denoted by $l = [c_l, r_l^1, r_l^2, ..., r_l^{|\mathcal{R}|}]$, where c_l and r_l^t, $t \in \mathcal{R}$ are respectively the cost and resource consumptions along the subpath corresponding to l. The resulting SPPRC is as follows :

$$(P_3) \quad \textit{Minimize} \quad \sum_{(i,j)\in A, i\geq i_0} c_{ij} x_{ij} + \sum_{i\in V, i\geq i_0, l\in\mathcal{S}_i} c_i^l y_i^l \tag{7.7}$$

s.t.

$$\sum_{i\geq i_0, l\in\mathcal{S}_i} y_i^l = 1 \tag{7.8}$$

$$\sum_{(i,j)\in Co_k} x_{ij} + \sum_{i>k, l\in\mathcal{S}_i} y_i^l = 1 \qquad \forall k \in \{i_0, i_0 + 1, ..., |V| - 1\} \tag{7.9}$$

$$y_i^l (r_l^t - R_i^t) \leq 0 \qquad \forall i \in V, i \geq i_0, \forall l \in \mathcal{S}_i, \forall t \in \mathcal{R} \tag{7.10}$$

$$x_{ij}(R_i^t + r_{ij}^t - R_j^t) \leq 0 \qquad \forall t \in \mathcal{R}, \forall (i, j) \in A, i \geq i_0 \tag{7.11}$$

$$a_i^k \leq R_i^t \leq b_i^t \qquad \forall t \in \mathcal{R}, \forall i \in V, i \geq i_0 \tag{7.12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A, i \geq i_0 \tag{7.13}$$

$$y_i^l \in \{0, 1\} \qquad \forall i \in V, i \geq i_0, \forall l \in \mathcal{S}_i \tag{7.14}$$

In this model, x_{ij} are the arc flow variables and y_i^l are the label variables that indicate whether or not label $l \in \mathcal{S}_i$ contributes to the construction of an optimal path. R_i^t, $t \in \mathcal{R}$, $i \in V$ are real variables that compute the resource consumptions along an optimal subpath from s to i.

Constraints (7.8) ensure that exactly one label is chosen to construct an optimal path. This may be the trivial label $l_1 = [0, 0, ..., 0] \in \mathcal{S}_1$ normally used by DP algorithms. Constraints (7.9) cover each cocycle Co_k using either a stored label $l \in \mathcal{S}_i, i \in V$, $i > k$ or an arc $(i, j) \in Co_k$. Constraints (7.10) and (7.11) are resource constraints that update the resource consumptions whenever a new arc or label is selected to be a part of an optimal path. Constraints (7.12) verify the feasibility of the path in terms of resource constraints. Finally, the binary requirements on variables x_{ij} and y_i^l are expressed by (7.13) and (7.14).

This model is a generalization of the classical formulation of the SPPRC that allows the construction of feasible paths using a completion of any label $l \in \cup_{i \in V} \mathcal{S}_i$ and not necessarily the trivial one $l_1 \in \mathcal{S}_1$. Furthermore, every nonempty selection of labels from $\cup_{i \in V} \mathcal{S}_i$ induces a subspace of the entire state space of the SPPRC. If these subspaces are disjoint and complementary, they provide a real decomposition of the whole state space.

MDDPA provides two rules for decomposing the state space. The first classifies stored labels according to their distance from the sink node. The distance of a given label $l \in \mathcal{S}_i, i \in V$ is measured in terms of the number of uncovered cocycles. Labels belonging to the same set of labels therefore have the same distance from the sink node, so the sets of stored labels $\mathcal{S}_i$, $i \in V$ form a decomposition of the state space. However, since there is no guarantee that the extension of these sets of labels will lead to feasible paths and since real-world networks have a huge number of nodes, we construct buckets of labels. Each bucket contains several sets of labels associated with a subset of sequential nodes in the topological order. These buckets are chosen to be disjoint and complementary, so they form a decomposition of the state space based on the distance criterion.

The second rule sorts the stored labels according to their reduced costs. Buckets of labels are constructed in such a way that each contains consecutive labels from a list ordered by reduced cost. Labels in the same bucket are associated with nodes of varying distances from the sink node and labels from the same node may now appear in different buckets. This gives a new decomposition of the state space based on reduced cost.

7.4.2.2 MDDPA search strategies

For each decomposition, MDDPA provides an appropriate search strategy to explore the induced subspaces. An *Nearest First* (NF) strategy is used for the decomposition based on the distance criterion, and a *Best First* (BF) strategy is used for the decomposition based on the reduced cost criterion.

The NF strategy first extends labels that require less computational effort to generate feasible paths, namely those for nodes relatively close to the sink node. Clearly, these labels have fewer uncovered cocycles than those for nodes far from the sink node. Consequently, they need fewer arcs to form complete paths, and hence less computational time. This strategy extends the buckets of labels one at a time in reverse topological order of the nodes. The BF strategy prioritizes the extension of the labels with the most negative reduced costs. This allows the most promising labels to be extended first, thus producing interesting paths as soon as possible. This strategy extends at each iteration the bucket that contains the most important labels in terms of reduced cost.

Algorithm 7.4 presents the pseudocode for MDDPA embedded in PCG. It provides a unified framework for the application of the NF and BF strategies. For illustration purposes and with no loss of generality, we denote by $\mathcal{P}_k, k \in \{1, 2, ..., K\}$ the sets of buckets of labels constructed using either of the two decompositions, where k is the index of these buckets. We denote by Ω^r the pool of negative-reduced-cost columns to send to the RMP in a given CG iteration r. The procedure for the RMP is denoted $RMP(\Omega^r, x^r, \alpha^r)$, where x^r and α^r are respectively the primal and dual solutions returned at iteration r.

At each MDDPA iteration k, we extend the labels by calling the $DP(G, \mathcal{L}, \Pi^k)$ procedure in the RSP induced by the bucket of labels $\mathcal{P}_k$. These labels are used to initialize the set of active labels $\mathcal{L}$. The set of feasible paths Π^k generated by $DP(G, \mathcal{L}, \Pi^k)$ is added to the pool of columns Ω^r. If the $CCP(\delta^r, \Omega^r, C^{best})$ returns *true*, the algorithm stops solving the SP, otherwise, a cost bounding is carried out using the $CostBounding(\Pi^k, C^{best})$ procedure before solving the next RSP. This procedure uses the reduced cost C^{best} of the least reduced cost path among the set of paths Ω^r found in the previous iterations to update the cost upper bounds in the nodes of the network.

Algorithm 7.4: Primal Column Generation using MDDPA

1: //Initialization //
2: Find an initial solution solution x^0, α^0
3: $\Omega \leftarrow \Omega^0$; $r \leftarrow 1$
4: **repeat**
5: //Solve the SP//
6: $\Omega^r \leftarrow \emptyset$; $C^{best} \leftarrow \infty$; $k \leftarrow 1$; $\mathcal{L}_1 \leftarrow \{l_1\}$; $\mathcal{L}_i \leftarrow \emptyset \ \forall i \in V \setminus \{1\}$
7: Update arc reduced costs using α^{r-1}
8: Construct $\bar{G}$ using the columns of the basic solution x^{r-1}
9: Run **LSP**$(\bar{G}, \mathcal{L}, \mathcal{S}, \Pi^0)$
10: Construct $\mathcal{P}_k, k \in \{1, 2, ..., K\}$ from $\mathcal{S}_i, i \in \{1, 2, ..., |V|\}$
11: **repeat**
12: $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \{l\} \ \forall i \in V \ \forall l$ such that $l \in \mathcal{P}_k \cap \mathcal{S}_i$
13: Run **DP**$(G, \mathcal{L}, \Pi^k)$
14: **if** $\Pi^k \neq \emptyset$ **then**
15: $\Omega^r \leftarrow \Omega^r \cup \Pi^k$
16: $\delta^r = z_{r-1} - z_r$
17: **if CCP**$(\delta^r, \Omega^r, C^{best}) =$ true **then**
18: **break**
19: **CostBounding**(Π^k, C^{best})
20: $k \leftarrow k + 1$
21: **until** $k = K$
22: //Solve the RMP//
23: **if** $\Omega^r \neq \emptyset$ **then**
24: $\Omega \leftarrow \Omega \cup \Omega^r$
25: Run **RMP**(Ω, x^r, α^r)
26: $r \leftarrow r + 1$
27: **until** $\Omega^r = \emptyset$
28: Return x^r

7.4.3 Primal adjacency-based algorithm

We first define the state space decomposition used by PAB and then explain how the algorithm works.

7.4.3.1 PAB decomposition

The PAB decomposition is based on the notion of *adjacency* between two paths, and between one path and a set of paths in the solution space of the SPPRC. Adjacency is a well-known linear programming notion, defined as follows :

Definition 17. *Let $\mathcal{P}$ be the polyhedron of a linear program. Two extreme points $\boldsymbol{x^1}$ and $\boldsymbol{x^2}$ of $\mathcal{P}$ are adjacent if there exists a face of $\mathcal{P}$ of dimension one (an edge) that contains both*

$\boldsymbol{x^1}$ *and* $\boldsymbol{x^2}$.

For the SPPRC, every feasible path corresponds to an extreme point of the solution space. Himmich *et al.* (2018b) have provided a new definition of adjacency between two paths in a network using the notion of a *detour* based on the notion of *compatibility*.

Definition 18. *A set of arcs $\mathcal{D}$ is said to be compatible with a path π if $\mathcal{D}$ is able to replace a subset of the arcs composing π to produce a new complete path π'.*

Definition 19. *Let π be a path. A set of arcs $\mathcal{D}$ is called a detour if it is compatible with π and minimal, in the sense that none of its strict subsets is compatible.*

Let π and π' be two paths and A^π and $A^{\pi'}$ their respective sets of arcs. The next proposition combines the notion of adjacency and detour.

Proposition 21. *(Himmich et al., 2018b) π' is adjacent to π if and only if there exists exactly one detour $\mathcal{D}$ such that $A^{\pi'} \setminus A^\pi = \mathcal{D}$.*

This proposition has been generalized in (Himmich *et al.*, 2018b) in two ways. The notion of *k-adjacency* is developed to refer to the degree of adjacency of a given path in relation to either another path or a set of paths. Hence, a path π' is said to be *k-adjacent* to a path π if and only if there are k different detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ allowing the construction of π' from π, i.e., $A^{\pi'} \setminus A^\pi = \cup_{i=1}^k \mathcal{D}_i$. Similarly, a path π is said to be *k-adjacent* to a set of paths Π if and only if there are k different detours $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that $A^\Pi \setminus A^\pi = \cup_{i=1}^k \mathcal{D}_i$ where A^Π is the set of arcs composing all the paths of Π.

Himmich *et al.* (2018b) proved that every set of paths coincides with a face in the solution space. The dimension of this face depends on the number of affinely independent elements in the set of paths. Consequently, given an initial set of paths Π^0, the degree of adjacency is a useful way to compute the distance of any path π to the face defined by Π^0. Additionally, this measure provides a new decomposition of the state space of the SPPRC. Each degree of adjacency has a corresponding subspace of the state space, and these subspaces are disjoint. This decomposition is the fundamental pillar of the PAB algorithm.

7.4.3.2 PAB algorithm

The PAB algorithm (hereafter referred to as PAB) is an combination of DP and the polyhedral properties described above. It solves the SPPRC using iterative calls of DP in restricted state subspaces related to different degrees of adjacency. From a polyhedral point of view, given an initial set of paths Π^0, PAB looks for negative-reduced-cost extreme points in the

neighborhood of the face corresponding to Π^0. This neighborhood is sequentially enlarged as the degree of adjacency is increased.

Recall that the degree of adjacency to Π^0 is the number of allowed detours from Π^0. Himmich *et al.* (2018b) added a new resource called the *adjacency resource*. This resource counts the number of times a subpath leaves Π^0, which fits with the number of detours. In a given iteration k, the upper bound of the adjacency resource is set to k at every node to prevent the extension of subpaths with a degree of adjacency greater than k.

We propose a simple method that avoids the use of the adjacency resource. In contrast to MDDPA, which uses LSP just once at the beginning of the process, PAB manages the flow of created labels using a dynamic call to LSP. The process is as follows : given an adjacency degree k and an initial set of paths Π^0, LSP performs a DP search in the subspace of degree k and returns a set of feasible paths that are k degrees distant from Π^0. Additionally, all labels of degree $k+1$ whose extension was stopped because of the adjacency degree are saved in the sets $\{\mathcal{S}_i, i \in V\}$. These labels are then extended at iteration $k+1$, which ensures a dynamic update of the set of saved labels.

Himmich *et al.* (2018b) proposed to initialize Π^0 with a set of artificial paths that start from the depot, cover a given trip, and return to the depot. Based on this structure of the initial point, the authors proposed to stop PAB once a maximum degree of adjacency is reached. This degree is shown to be the maximum degree of adjacency that a feasible path may have in relation to Π^0.

In this paper, we propose to construct the initial point Π^0 using the paths corresponding to the nondegenerate basic columns extracted from the basic solution of the RMP. These paths have zero reduced costs, so minor changes may be able to produce the desired negative-reduced-cost paths. Such useful changes are exactly what PAB aims to find using detours. So the subnetwork $\bar{G}$ is composed of the nodes and arcs composing the paths of Π^0 as explained in Section 7.4.2.1.

Furthermore, we propose a more general stopping criterion that does not depend on the structure of the initial point. Instead of stopping on a predefined maximum degree of adjacency, our implementation increases the degree of adjacency sequentially and stops whenever the sets of stored labels are empty. The next proposition verifies the accuracy of the algorithm with this stopping criterion.

Proposition 22. *For a PCG iteration, if $\mathcal{S} = \emptyset$, the solution returned by PAB is optimal for the SP.*

Proof. If $\mathcal{S} = \emptyset$, this means that there is no label whose extension has been prevented by

PAB because of violation of the degree of adjacency. Thus, the degree of adjacency is no longer restrictive. Consequently, PAB becomes similar to the DP algorithm, and the solution of the final iteration is optimal. □

Algorithm 7.5 presents the pseudocode for PAB embedded in PCG.

Algorithm 7.5: Primal Column Generation using PAB

```
1: //Initialization //
2: Find an initial solution solution x^0, α^0
3: Ω ← Ω^0 ; r ← 1
4: repeat
5:    //Solve the SP//
6:    Ω^r ← ∅ ; C^best ← ∞ ; k ← 1 ; L_1 ← {l_1} ; L_i ← ∅ ∀i ∈ V \ {1}
7:    Update arc reduced costs using α^{r−1}
8:    Construct Ḡ using the columns of the basic solution x^{r−1}
9:    repeat
10:      Run LSP(Ḡ, L, S, Π^k)
11:      if Π^k ≠ ∅ then
12:         Ω^r ← Ω^r ∪ Π^k
13:         δ^r = z_{r−1} − z_r
14:         if CCP(δ^r, Ω^r, C^best) = 1 then
15:            break
16:         CostBounding(Π^k, C^best)
17:      L ← S
18:      k ← k + 1
19:   until S = ∅
20:   //Solve the RMP//
21:   if Ω^r ≠ ∅ then
22:      Ω ← Ω ∪ Ω^r
23:      Run RMP(Ω, x^r, α^r)
24:      r ← r + 1
25: until Ω^r = ∅
26: Return x^r
```

Remark 5. *No label is generated more than once by PAB.*

Remark 5 shows that no redundant work is done by PAB, so no subspace is invoked more than once. Moreover, similarly to MDDPA, the best reduced cost found in a given PAB iteration is used to fathom nonpromising labels to tighten the subsequent subspaces. These two features allow the algorithm to reduce the complexity of DP by reducing the number of created labels.

7.5 Computational experiments

In this section, we assess the usefulness of our PCG framework, carrying out a computational study where we solve the linear relaxation of VCSP instances. We compare the standard CG algorithm to the new PCG framework using MDDPA with the NF strategy, MDDPA with the BF strategy, and PAB. We begin by describing our test instances.

7.5.1 Test instances

The VCSP test instances correspond to acyclic networks and were randomly generated using the VCSP generator described in Haase *et al.* (2001). The complexity of a VCSP depends on the size of the instance, the number of resources, and the width of the resource intervals.

The size of an instance is measured in terms of the number of d-trips to cover, i.e., $tr(rp+1)$, where tr is the number of trips and rp is the number of relief points per trip. We set the number of trips to 120, 160, 200, or 240, and there are five or seven relief points. Each pair (rp, tr) leads to an instance type denoted rp_tr. We generated five instances of each type by varying the seed number, for a total of 40 instances. We classify them into 5_rp and 7_rp instances (see Table 7.1).

Table 7.1 List of test instances (PCG)

5_rp instances				
Type	No.	nodes	arcs	d-trips
5_120	1	50334	78493	720
	2	55667	86437	720
	3	47906	74815	720
	4	51661	80459	720
	5	47882	74742	720
5_160	1	92251	142502	960
	2	97623	150501	960
	3	88244	136442	960
	4	92235	142451	960
	5	86939	134534	960
5_200	1	144942	219962	1200
	2	154597	237237	1200
	3	136573	210147	1200
	4	142169	218646	1200
	5	139090	214042	1200
5_240	1	211558	323950	1440
	2	218536	334388	1440
	3	194907	299016	1440
	4	202792	310874	1440
	5	199094	305235	1440

7_rp instances				
Type	No.	nodes	arcs	d-trips
7_120	1	98395	151488	960
	2	109052	167422	960
	3	92990	143343	960
	4	100960	155311	960
	5	93059	143398	960
7_160	1	180189	275631	1280
	2	191268	292192	1280
	3	173005	264803	1280
	4	180803	276516	1280
	5	169533	259635	1280
7_200	1	280186	427164	1600
	2	304154	463103	1600
	3	267271	407720	1600
	4	279205	425726	1600
	5	272429	415569	1600
7_240	1	414189	629747	1920
	2	429705	652996	1920
	3	381497	580745	1920
	4	398211	605857	1920
	5	389186	592206	1920

We consider seven resource constraints : the minimum and maximum number of pieces of work in the duty, the minimum and maximum length of each piece of work, the length of the duty, the length of breaks, and the total work time in the duty, i.e., the time spent driving or waiting for a bus. Note that, in order to make the problem more difficult, we consider duties with up to two or four pieces of work while only a maximum number of one or two pieces of work per duty was considered in Haase *et al.* (2001). The lower and upper bounds of the resource constraints are given in Table 7.2, which is reproduced from Haase *et al.* (2001).

Table 7.2 Work rules for a driver schedule

	Minimum	Maximum
No. of pieces	1	2 or 4
Piece length (min)	15	300
Duty length (min)	45	600
Work time (min)	30	480
Break time (min)	15	90

The experiments were conducted on a computer with an Intel Core i7 3.40 Ghz processor and 16 GB of memory running LINUX. Our PCG framework is implemented in C++, and the LP solver is IBM ILOG CPLEX 12.8.0.0. To solve the SPs, standard DP, MDDPA and PAB were all implemented in C++ using Boost Graph library, a well-known C++ library. In addition, we have fairly enhanced these algorithms, as a preprocessing step, with a topological order of the nodes, and well tightened cost and resource upper bounds for all the nodes of the network. Finally, for our three proposed algorithms, we have fairly considered the following values of the CCP parameters : $\bar{\delta} = 10^{-3}$, $\bar{C} = -10^{-5}$.

7.5.2 Computational results

The computational results are reported in Tables 7.3 and 7.4 for instances with five and seven relief points, respectively. For each instance type (Column 1), the test instances are numbered from one to five in Column 2. Then, for each algorithm and each instance, we give the number of CG iterations (Itr.), the total time spent solving the SPs (SP time), the total solution time (T time), and finally the total number of columns generated (Col.) ; all times are in seconds. We refer to the standard CG algorithm as stdCG and to the PCG algorithms as PCG-NF, PCG-BF, and PCG-PAB.

Table 7.3 Computational times for 5_*rp* instances

Instance		stdCG				PCG-NF				PCG-BF				PCG-PAB			
Type	No.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.
5_120	1	167	1347.0	1498.0	40083	145	419.9	605.2	39848	119	354.1	502.1	26067	490	414.7	628.3	26380
	2	242	835.3	950.9	40445	262	472.4	673.0	44891	298	528.8	692.0	35204	518	567.0	736.3	24988
	3	155	768.2	871.7	37674	121	261.0	395.1	36808	110	236.1	328.6	26069	382	264.7	418.3	21802
	4	187	1260.1	1392.4	40125	195	438.3	651.4	43583	161	429.6	600.9	32863	510	592.2	805.0	25090
	5	286	2008.0	2309.0	50012	201	665.6	944.7	48580	199	829.7	1106.5	42893	665	826.2	1180.3	32050
5_160	1	258	6665.6	7236.1	60388	231	1299.4	2263.7	68240	181	1344.7	1938.5	41665	699	1361.7	2282.1	38932
	2	358	3968.2	4439.8	75705	314	1686.1	2272.0	64844	317	1758.1	2372.7	57118	823	1859.9	2580.8	44887
	3	242	3960.5	4419.4	59324	195	1372.9	2007.0	63333	189	1093.5	1507.5	43029	633	1047.2	1587.8	34594
	4	418	12985.3	13946.9	80177	336	4798.5	5806.4	76893	287	3331.1	4083.6	53774	861	3296.4	4693.9	42517
	5	372	8574.5	9770.6	75757	306	3233.0	4788.7	90342	310	2624.0	3795.9	57786	985	2806.8	4240.9	48157
5_200	1	299	17187.1	18966.2	90377	221	3203.8	5359.9	83400	194	3794.4	4931.8	55310	968	3019.6	5899.4	58303
	2	450	20502.6	22213.6	97862	424	7304.4	9930.0	107409	393	7532.0	9309.8	73956	1221	6643.8	9228.7	63897
	3	298	11227.5	12271.6	84270	211	2237.5	3737.0	82224	208	2751.3	3811.0	60437	817	2088.9	3769.5	49298
	4	322	14972.1	16633.3	94451	250	2744.6	5543.3	87144	232	2937.8	4585.9	61106	857	2449.2	4603.2	50269
	5	420	28301.2	31154.3	99950	465	16217.1	19353.2	107408	408	12765.5	15407.7	78996	1280	9457.1	13558.5	61841
5_240	1	479	91559.3	96529.7	126701	393	17073.1	21755.9	116714	324	22385.9	25891.8	89556	1368	10891.3	18514.5	76347
	2	637	46902.7	50990.1	129190	423	9121.8	13629.6	114619	426	11140.7	14589.4	86229	1358	8783.9	14284.1	75236
	3	396	34909.9	39033.4	123348	299	6777.5	11753.0	119501	272	6681.5	10467.8	80513	1278	6327.2	12210.0	71790
	4	289	13134.3	15728.4	96234	240	3842.8	7236.0	99949	211	3913.0	5864.1	66070	896	3283.8	6747.1	61921
	5	282	37346.1	38623.8	132222	268	23206.3	24787.0	137447	223	15842.0	17162.3	104168	973	6838.1	8516.5	88295

For the 40 instances considered, the PCG framework was faster than stdCG. The primal algorithms greatly reduced both the SP time and the total time. To show this, we compute the total time reduction factors as a ratio of the total stdCG time to the total PCG time using each of the three primal algorithms. For the 5_*rp* instances, PCG-NF reduced the total time by a factor of between 1.41 to 4.44, with an average ratio of 2.57. For PCG-BF, these ratios range between 1.37 and 3.85 with an average of 2.85. Similarly, PCG-PAB reduces the total time by a factor of between 1.29 and 5.21, with an average of 2.80. For the 7_*rp* instances, PCG-NF reduced the total time by a factor of up to 6.18, with an average of 3.07. The ratio is about 3.29 on average for PCG-BF, and PCG-PAB was slightly more efficient with an average ratio of 3.40.

These improvements are mainly due to the huge reductions in the SP times. To clarify this, we define the SP time reduction factor for each PCG algorithm as the ratio of the SP time for this algorithm to the SP time for stdCG. For the 5_*rp* instances, PCG-NF and PCG-BF give an overall average SP time reduction factor of 3.53 and about 3.49 respectively. For PCG-PAB, these factors are between 1.47 and 8.41, with an overall average of 4.10. The values grow significantly with problem size. For the 7_*rp* instances, PCG-NF reduces the SP time by a factor of up to 7.13, with an average of 3.82, while PCG-BF has an average reduction factor of 3.85. For PCG-PAB, the factor reaches 9.69, with an overall average of 5.46. Figure 7.2 gives, for each instance type, the SP time reduction factors realized by our three primal algorithms in a PCG framework. Each factor is computed as the average of the

SP times for the five instances of this type.

Table 7.4 Computational times for 7_*rp* instances

Instance		stdCG				PCG-NF				PCG-BF				PCG-PAB			
Type	No.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.	Itr.	SP time	T time	Col.
7_120	1	256	7676.4	8463.4	69686	186	1661.1	2512.3	67002	138	1306.8	1745.3	39141	693	1540.4	2408.9	40495
	2	336	4022.1	4421.1	63965	326	1771.7	2564.4	70352	316	1838.4	2423.2	54368	751	2190.4	2780.8	41247
	3	219	2950.5	3410.4	61852	124	779.7	1139.0	55375	132	626.4	877.7	34747	481	697.8	1191.2	30402
	4	269	6409.9	6887.4	67347	195	1369.5	2012.3	60706	187	1826.5	2333.8	46434	685	1551.8	2105.3	39050
	5	314	6966.7	8083.0	77798	249	2305.9	3085.9	68698	216	2172.6	2848.1	51268	892	2991.7	4478.9	49951
7_160	1	329	29108.3	31622.3	99634	252	5320.1	8728.6	103850	201	4918.3	6277.1	57058	924	3605.3	6553.3	55805
	2	544	26181.3	28299.4	126027	324	7863.0	9804.4	97071	336	5619.4	7710.6	78268	1074	5419.9	8243.0	67497
	3	287	14921.7	16588.1	92484	225	4848.1	6847.4	87344	196	3410.2	4677.6	55074	743	2631.3	4122.7	49566
	4	524	52612.6	57084.2	126860	351	12714.2	16824.3	111255	310	13136.0	16961.4	81371	1234	10535.8	15994.9	66880
	5	401	28761.5	33176.3	115099	342	12077.2	16622.7	125054	324	10348.7	13572.6	86999	1234	7625.5	13445.6	68221
7_200	1	317	66294.5	72890.8	130037	281	13888.2	19435.8	127378	209	12794.2	15669.1	76588	1166	8024.4	18021.4	84573
	2	710	134298.0	145517.0	174340	442	30803.6	39477.8	136850	446	34301.5	41871.0	112559	1574	24229.1	36536.6	94121
	3	335	35169.1	39508.8	126936	233	9807.4	13356.2	122581	238	10266.1	12645.4	79869	1080	6930.0	14619.0	76004
	4	334	42611.6	48253.5	135293	294	10647.4	15793.1	120517	242	8282.0	11534.0	81069	1194	7675.2	18422.0	81578
	5	427	102086.0	113160.0	146153	451	54087.4	64479.1	133504	441	61017.5	69857.6	119576	1473	17825.1	34309.4	90534
7_240	1	482	499539.0	518601.0	182595	401	70083.4	83920.9	165695	350	131481.0	142005.0	122630	1618	76686.7	106170.0	107763
	2	710	279066.0	300450.0	199696	478	71804.8	84481.9	158733	522	77824.7	90910.6	131615	1667	28793.6	49037.2	111352
	3	399	105834.0	124081.0	177159	338	32000.0	46618.1	178402	359	49288.8	60045.4	122245	1542	15362.4	37954.8	102502
	4	296	42070.1	48890.2	136887	262	18002.7	25230.4	136910	247	18711.5	23253.2	89473	1165	10167.6	20651.9	89363
	5	323	82134.1	96630.1	125825	256	18753.9	28972.3	113443	226	17809.6	29384.7	80095	1703	11713.4	29507.9	82245

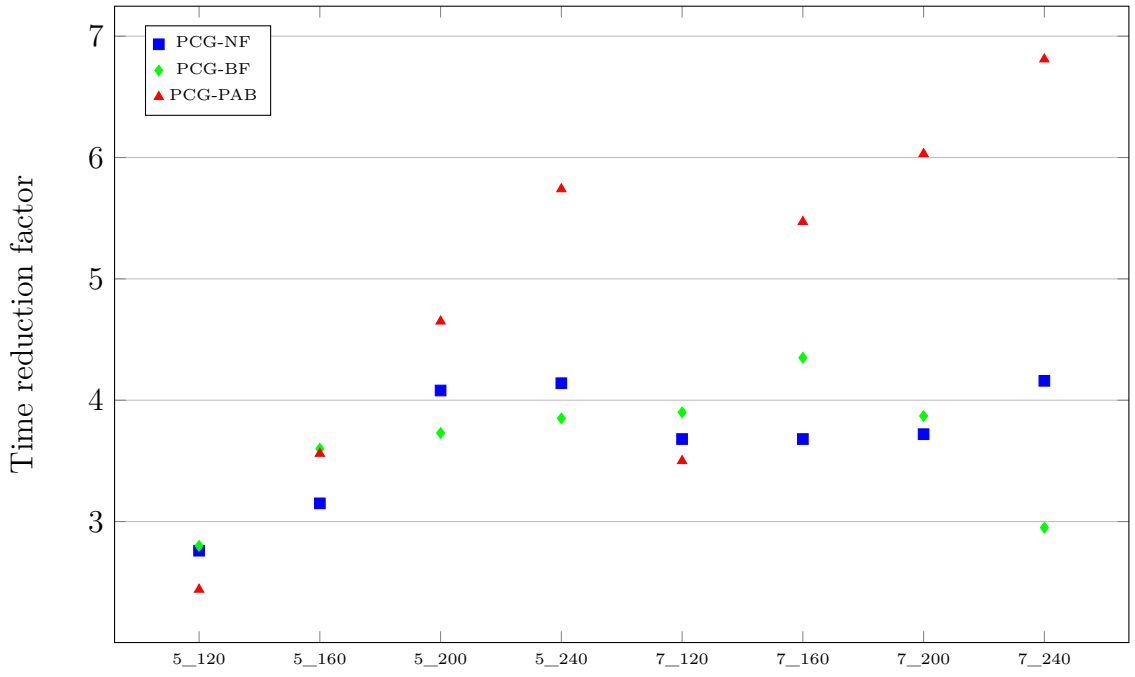

Figure 7.2 SP time reduction factors

Another important feature of the PCG framework is the substantial reduction in the SP

time as a percentage of the total time. Tables 7.3 and 7.4 show that the SP times often represent more than 85% of the total time (the average is about 90%) when stdCG is used. The average value is about 70% for PCG-NF, 76% for PCG-BF, and does not exceed 65% for PCG-PAB. This is partially explained by increased MP times, but the increases are not significant compared to the huge reductions in the SP times.

PCG also reduces the number of generated columns. This reduction was not significant for PCG-NF, but it appears clearly with the other two algorithms. PCG-BF reduces the number of columns by 27% on average for 5_*rp* instances and 34% on average for the 7_*rp* instances. PCG-PAB generates on average at most 60% of the number of columns generated by stdCG.

Furthermore, Tables 7.3 and 7.4 show that MDDPA reduces the number of CG iterations for almost all the instances. For the 5_*rp* instances, PCG-NF reduces the number of CG iterations by an average of 15% and PCG-BF by 22%. For the 7_*rp* instances these values are 23% and 28%. However, PAB increases the number of CG iterations by, on average, a factor of 3.5.

Table 7.5 Average SP and MP times per iteration (in seconds)

Instance type	MP time per iteration				SP time per iteration			
	stdCG	NF	BF	PAB	stdCG	NF	BF	PAB
5_120	0.7	1.1	1.0	0.4	6.0	2.5	2.7	1.0
5_160	2.1	3.4	2.7	1.2	21.5	8.6	7.8	2.5
5_200	5.0	8.1	5.7	2.5	50.9	17.6	19.2	4.3
5_240	9.4	11.7	9.5	3.9	106.2	38.1	41.9	6.0
Average	4.3	6.1	4.7	2.0	46.2	16.7	17.9	3.5
7_120	2.3	3.2	2.5	1.1	20.3	7.4	8.0	2.4
7_160	7.3	10.6	8.2	3.3	72.1	27.7	26.6	5.4
7_200	18.3	18.9	14.7	8.6	174.0	63.5	70.8	9.4
7_240	46.8	34.0	30.7	12.7	418.2	112.3	163.3	18.0
Average	18.7	16.7	14.0	6.4	171.2	52.7	67.2	8.8

Table 7.5 gives the average MP time and SP time per iteration for each instance type. This table reveals an important characteristic of PCG-PAB : its ability to perform small improvements in less time than stdCG, PCG-NF and PCG-BF. For the 5_*rp* instances, the average SP time per iteration for PCG-PAB is about 13 times less than that for stdCG and five times less than that for both PCG-NF and PCG-BF. These reduction factors are larger for the 7_*rp* instances : the SP time per iteration is reduced by average factors of 19, 6, and 7 compared to stdCG, PCG-NF, and PCG-BF respectively. Moreover, PCG-PAB reduced the MP time per iteration of stdCG by a factor of 2 for the 5_*rp* instances and about 3 for the 7_*rp* instances.

PCG-PAB focuses its search on small neighborhoods of the current basic solution. This allows it to quickly find good negative-reduced-cost columns without investing a large computational effort in the SPs. Consequently, it sends only a few columns to the MP. The dual values are then rapidly updated, which ensures fast convergence of the overall PCG algorithm in a larger number of iterations. The PCG-NF and PCG-BF algorithms are also primal, but they require longer computational times to return good columns because they explore larger subspaces. In summary, PCG-PAB is faster but with small improvements, while PCG-NF and PCG-BF are slower but with larger improvements. This can be clearly seen in Figure 7.3.

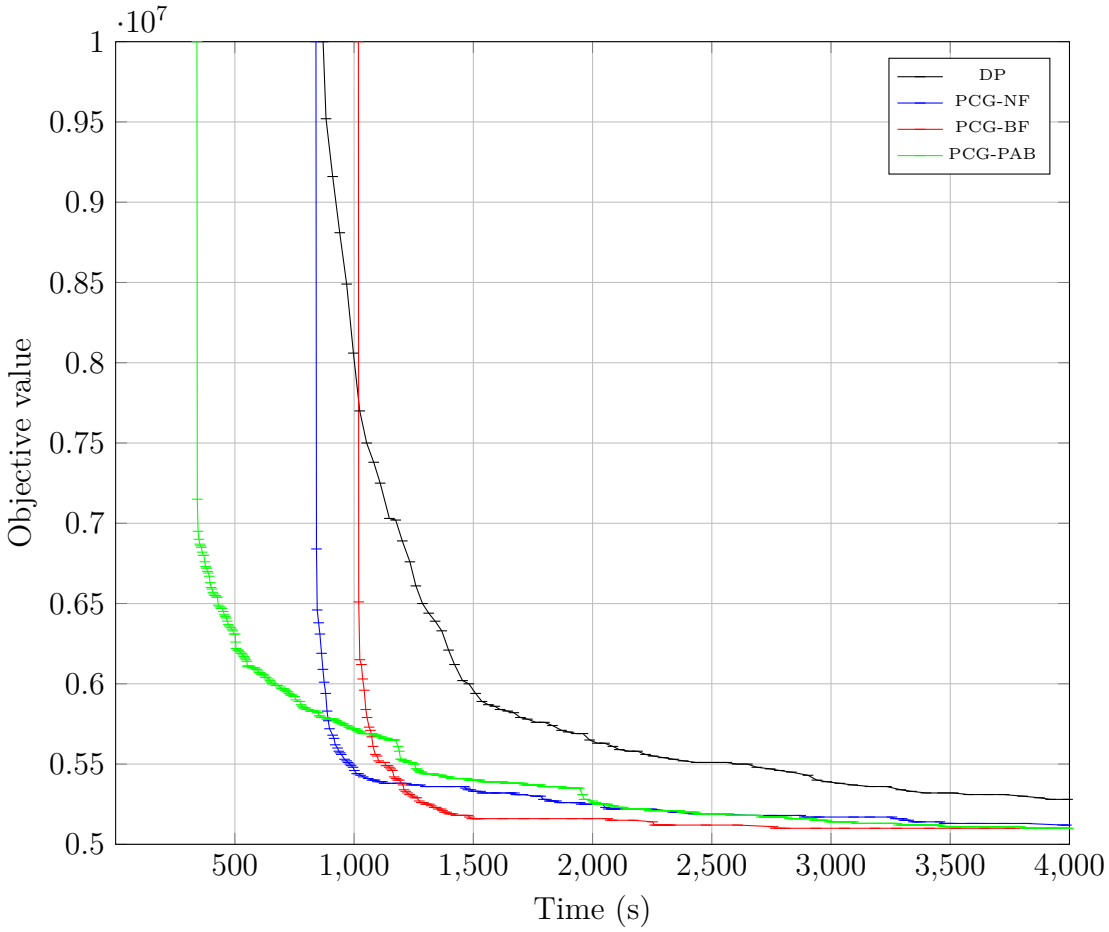

Figure 7.3 Improvement of the objective value over time

Figure 7.3 is based on data from one 7_*rp* instance; it tracks the evolution of the objective value (cost) over time for all the algorithms. For clarity, it focuses on the region with the most important differences between the algorithms. For the selected instance, the total time reduction factor was 4.04 for PCG-PAB, 3.75 for PCG-NF, and 4.65 for PCG-BF.

PCG outperforms stdCG in terms of the speed of convergence, regardless of the algorithm used to solve the SPs. In particular, until 1500 s, the cost achieved by stdCG is at least three times greater than the objective value reached by any of our algorithms. This ratio then decreases until it becomes approximately 2. Moreover, the graph shows that PCG-NF and PCG-BF decrease the objective value in a similar way. PCG-NF is initially faster, and then PCG-BF becomes more efficient. PCG-PAB gives a slow and continuous decrease of the objective value, and the decrease starts earlier than that for PCG-NF and PCG-BF.

PCG-PAB makes small cost decreases early, providing the best solution quality at this stage. It is then overtaken by PCG-NF and, later, PCG-BF. PCG-PAB is able to generate good columns at the beginning of the CG because it performs an adjacency-based search in the neighborhood of the current basic solution. PCG-NF and PCG-BF have better behavior once the dual values start to stabilize. For all the algorithms, the cost decrease is slow toward the end of the process.

To conclude this section, we observe that the PCG framework is remarkably efficient compared to the classical CG framework. The time reduction factors are large, and the reductions in the number of columns are also considerable, especially for PCG-PAB. PAB is initially the best algorithm, and later the NF and BF strategies may be alternated within MDDPA.

7.6 Conclusion

In this paper, we have provided a new PCG framework for the VCSPs. This work is the first attempt to embed into a CG scheme recently developed primal algorithms for the SPs, namely PCG-PAB, PCG-NF, and PCG-BF. We focused on the skeleton of the PCG framework, the underlying primal paradigm, and the state space decompositions used. Furthermore, we adapted the new primal methods to the CG context by introducing heuristic and a valid stopping criteria that allow the CG to quickly and intelligently find negative-reduced-cost columns. The PCG is an innovative framework for CG algorithms, it gives the SP solver a large degree of flexibility and autonomy, which improves the overall performance. Our experiments show that the PCG framework outperforms standard CG, producing optimal solutions for all the instances in shorter computational times. In particular, the SP times were reduced by factors of up to 7.

We plan to apply this work to airline crew scheduling problems, where the SPPRC arises as a pricing SP inside CG. We also plan to investigate the strengths of the three primal methods in order to find a way to combine them in a CG process. Finally, our long-term goal is to develop efficient new learning techniques that could benefit from the flexibility of our PCG

framework and help the solver focus on the regions in the SP state spaces with the most potential.

Acknowledgements

This work was supported by a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Kronos Inc. The authors would like to thank these organizations for their support and confidence.

CHAPITRE 8 DISCUSSION GÉNÉRALE

Dans cette thèse, nous avons traité le SPPRC, un problème qui apparaît essentiellement comme SP lors de la résolution des problèmes de tournées de véhicules et d'horaires d'équipages avec la méthode de CG. Étant donné son exactitude, sa capacité de manipuler les contraintes de ressources complexes, et sa capacité de générer plusieurs solutions entières réalisables au lieu d'une seule, la DP est souvent considérée comme étant une méthode idéale pour résoudre le SPPRC. Toutefois, comme toute méthode exhaustive, la DP perd son efficacité face aux problèmes de grande taille.

L'objectif principal de ce travail était de faire face aux faiblesses de la DP qui relèvent principalement du problème de la dimensionnalité. Le défi que nous avons relevé était alors de proposer de nouvelles méthodes de résolution du SPPRC qui devraient être à la fois exactes, efficaces par rapport à la DP, tirant profit des avantages de cette dernière et adaptées aux besoins de la CG.

Pour ce faire, nous avons opté pour des approches primales pour deux raisons : premièrement, parce que ces approches traitent les problèmes d'optimisation de façon graduelle, ce qui permet d'alléger le problème de dimensionnalité dont souffre la DP ; et deuxièmement, car ces méthodes ont l'avantage de produire des solutions primales réalisables au cours du processus de résolution, ce qui permet d'arrêter ce dernier une fois que des solutions satisfaisantes sont trouvées, une qualité largement souhaitable dans un contexte de CG.

C'est dans cette optique primale que nous avons proposé dans un premier volet de ce travail l'algorithme MDDPA. Il s'agit d'un algorithme primal exact combinant trois techniques qui fonctionnent ensemble pour résoudre le problème de la dimensionnalité et surmonter les faiblesses des algorithmes DP. L'algorithme effectue une partition disjointe de l'espace d'états en utilisant une procédure de stockage d'étiquettes. MDDPA utilise ensuite deux stratégies de chargement d'étiquettes pour l'exploration itérative d'une séquence de sous-espaces d'états restreints. Ces deux techniques permettent la construction d'un outil primal puissant permettant à l'itération courante de tirer profit des précédentes. De plus, la procédure LLEL permet de détecter des directions de descente qui aident le solveur à trouver de nouveaux chemins en investissant un effort minimal. Nous avons évalué l'algorithme MDDPA sur des instances de grande taille de VCSP extraites de différentes étapes du processus de CG. L'algorithme a prouvé son efficacité par rapport à la DP aussi bien au début qu'à la fin de la CG. En particulier, il est capable de générer des chemins avec plus de 90% du coût optimal en moins de 10% du temps total requis par la DP.

Les résultats étonnants obtenus par MDDPA nous ont encouragé par la suite à poursuivre la recherche dans la même direction des algorithmes de type primal. C'est ainsi que nous avons décidé, dans un deuxième volet de ce travail, d'entamer une étude polyédrique approfondie de l'espace des solutions du SPPRC. Notre étude nous a permis d'extrapoler certaines propriétés polyédriques du problème. Afin de mettre en valeur ces résultats, nous avons proposé l'algorithme PAB qui effectue une exploration itérative de l'espace d'états avant de converger à une solution optimale. En particulier, nous utilisons la notion d'adjacence par rapport à un point initial afin de restreindre l'espace d'états à des sous-espaces de taille réduite. De plus, en utilisant le concept de combinaisons affines, nous avons montré que de meilleurs chemins peuvent être facilement générés en combinant des chemins existants.

Nous avons évalué notre méthode sur des instances de VCSP issues de la littérature. Une comparaison avec la méthode de DP a montré la performance supérieure de PAB. En effet, cet algorithme a pu réduire le temps de résolution pour toutes les instances d'un facteur de réduction variant entre 2 et 5 en moyenne. Le PAB converge vers des solutions optimales plus rapidement que MDDPA, et prouve leur optimalité tout en restant compétitif à MDDPA. De plus, le PAB produit des chemins réalisables de meilleure qualité dans des délais très courts.

Les résultats obtenus ont montré clairement que nos deux algorithmes MDDPA et PAB répondent largement aux critères fixés au début de notre projet de recherche. D'abord, il s'agit de méthodes exactes. Ensuite, ce sont des méthodes très efficaces par rapport à la DP aussi bien sur le plan de l'effort de calcul investi, que sur le plan du temps de résolution. De plus, ces méthodes sont équipées d'une variété de techniques d'apprentissage qui les rendent capables de tirer profit de l'information primale disponible afin d'accélérer le processus de résolution. Finalement, l'aspect primal de ces algorithmes permet un arrêt prématuré du processus de résolution lorsque la qualité des colonnes est jugée suffisante. Théoriquement, ces attributs font de MDDPA et PAB des algorithmes adéquatement adaptés à la méthode de CG. Afin de vérifier expérimentalement cette assertion, nous avons décidé d'évaluer nos deux méthodes dans un contexte de CG. C'est ainsi que nous avons proposé, dans un troisième volet de ce travail, un nouveau cadre de résolution utilisant la CG, mais adapté à nos méthodes primales.

Un des principaux objectifs de ce travail était d'établir un squelette du nouveau cadre de résolution PCG, le paradigme primal derrière celui-ci et les différentes décompositions de l'espace d'états utilisées à l'intérieur. De plus, nous avons adapté les nouvelles méthodes primales proposées au contexte de CG en introduisant des critères d'arrêt heuristiques et exacts. Ces améliorations permettent au solveur de trouver rapidement et intelligemment les colonnes de coûts réduits négatifs nécessaires. Nos tests expérimentaux sur des instances du

VCSP ont montré que le cadre PCG est plus performant que la CG standard couramment utilisée dans la littérature. Le PCG a pu produire des solutions optimales pour toutes les instances dans des temps de calcul très réduits. Ces performances en matière de temps total de résolution étaient principalement dues aux énormes réductions réalisées au niveau des temps de résolution des SPs. Ces aboutissements justifient expérimentalement l'efficacité des méthodes primales proposées pour résoudre les SPs, et confirment d'ailleurs l'importance de la piste de recherche explorée dans cette thèse.

CHAPITRE 9 CONCLUSION ET RECOMMANDATIONS

La recherche effectuée dans le cadre de cette thèse a été fructueuse dans le sens que nous avons développé trois nouveaux algorithmes efficaces pour résoudre le problème de plus court chemin avec contraintes de ressources. L'intégration de ces méthodes dans un contexte de génération de colonnes a donné naissance à un outil de résolution puissant pour les problèmes d'horaires d'équipages. Les performances expérimentales de ces algorithmes valident les contributions théoriques fournies, confirment la fiabilité des décompositions de l'espace d'états proposées et mettent en valeur le paradigme primal introduit dans cette thèse.

Ce paradigme ouvre de nouvelles pistes de recherche de grande envergure. Nous envisageons d'abord la solution efficace des problèmes de planification des compagnies aériennes, où le problème de plus court chemin avec contraintes de ressources se présente comme un sous-problème au sein de la génération de colonnes. Une autre perspective serait d'étendre la méthodologie proposée à d'autres problèmes d'optimisation combinatoire qui font appel à la programmation dynamique, et pour lesquels il est possible de partitionner l'espace de recherche en plusieurs sous-espaces.

Sur le plan de l'amélioration des méthodes proposées, il serait intéressant d'explorer la force de chacune des méthodes primales proposées afin de trouver un moyen de les alterner dans la même exécution de la génération de colonnes primale proposée. Dans la même perspective, un objectif à long terme de cette recherche serait probablement de développer de nouvelles techniques d'apprentissage efficaces susceptibles de tirer profit de la flexibilité de la structure de notre méthode de génération de colonnes primale. Ces techniques peuvent potentiellement guider le solveur afin d'intensifier la recherche dans les régions les plus prometteuses de l'espace d'états du sous-problème. Une autre extension du présent travail serait de considérer des fonctions de prolongation non linéaires qui apparaissent dans un bon nombre d'applications industrielles. Finalement, il serait également intéressant de développer un cadre de résolution parallèle du sous-problème. Cette piste semble être très prometteuse vu la structure disjointe des sous-problèmes restreints provenant des décompositions considérées.

RÉFÉRENCES

Y.P. Aneja, V. Aggarwal, et K.P.K. Nair (1983). Shortest chain subject to side constraints. *Networks*, *13(2)*, 295–302.

M. Ball, L. Bodin, et R. Dial (1983). A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science*, *17*, 4–31.

C. Barnhart, N. Boland, L.W. Clarke, E.L. Johnson, G.L. Nemhauser , et R.G. Shenoi (1998). Flight string models for aircraft fleeting and routing. *Transportation Science*, *32*, 208–220.

J.E. Beasley et N. Christofides (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19(4)*, 379–394.

S. Bunte et N. Kliewer (2009). An overview on vehicle scheduling models. *Public Transport*, *1(4)*, 299–317.

E.A. Cabral (2005). *Wide area telecommunication network design : Problems and solution algorithms with application to the Alberta supernet*. Thèse de doctorat, University of Alberta, Edmonton, Alberta, Canada.

W.N. Carlyle, J.O. Royset, et R.K. Wood (2008). Lagrangean relaxation and enumeration for solving constrained shortest-path problems. *Networks*, *52*, 256–270.

G.B. Dantzig et P. Wolfe (1960). Decomposition principle for linear programs. *Operations Research*, *8(1)*, 1–157.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marce, B. Rioux, M.M. Solomon, et F. Soumis (1997). Crew pairing at Air France. *European Journal of Operational Research*, *97(2)*, 245–259.

G. Desaulniers, J. Desrosiers, M. Gamache, et F. Soumis (1998a). Crew scheduling in air transportation. *Fleet Management and Logistics*, Part of the Centre for Research on Transportation book series (CRT), T. Crainic, G. Laporte (eds.), Kluwer. 169–185.

G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, et D. Villeneuve (1998b). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. *Fleet Management and Logistics*, Part of the Centre for Research on Transportation book series (CRT), T. Crainic, G. Laporte (eds.), Kluwer. 57–93.

G. Desaulniers et D. Villeneuve (2000). The shortest path problem with time windows and linear waiting costs. *Transportation Science*, *34(3)*, 312–319.

M. Desrochers (1986). *La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes*. Thèse de doctorat, Université de Montréal, Montreal, Canada.

M. Desrochers, J. Desrosiers, et M. Solomon (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*, 342–354.

M. Desrochers et F. Soumis (1988a). A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, *26*, 191–212.

M. Desrochers et F. Soumis (1988b). A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, *35*, 242–254.

M. Desrochers et F. Soumis (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, *23*, 1–13.

J. Desrosiers, Y. Dumas, M.M. Solomon, et F. Soumis (1995). Time constrained routing and scheduling. *Network Routing, Handbook in Operations Research and Management Science, Elsevier Science, Amsterdam*, *8*, 35–139.

J. Desrosiers, Y. Dumas, et F. Soumis (2003). The multiple vehicle dial-a-ride problem. *Computer-Aided Transit Scheduling*, Part of the Lecture Note in Economics Mathematical Systems book series (LNE, volume 308), J.R. Daduna, A. Wren (eds), Springer-Verlag, Berlin, Germany. 15–27.

J. Desrosiers, P. Pelletier, et F. Soumis (1983). Plus court chemin avec contraintes d'horaires. *RAIRO Recherche Opérationnelle*, *17(4)*, 357–377.

J. Desrosiers, F. Soumis, et M. Desrochers (1984). Routing with time windows by column generation. *Networks*, *14(4)*, 545–565.

I. Dumitrescu et N. Boland (2003). Improved preprocessing, labeling and scaling algorithm for the weight-constrained shortest path problem. *Networks*, *42(3)*, 135–153.

D. Feillet, P. Dejax, M. Gendreau, et C. Gueguen (2004). An exact algorithm for the elementary shortest path problem with resource constraints. *Networks*, *44*, 216–229.

D. Feillet, M. Gendreau, et L.-M. Rousseau (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, *45(4)*, 239–256.

M. Fischetti, S. Martello, et P. Toth (1989). The fixed job scheduling problem with working-time constraints. *Operations Research*, *37*, 395–403.

R. Freling, G. Boender, et J.M.P. Paixao (1995). An integrated approach to vehicle and crew scheduling. *Technical Report 9503-A, Economy Institute, Erasmus University, Rotterdam.*

R. Freling, D. Huisman, et A.P.M. Wagelmans (2003). Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, *6*, 63–85.

M.R. Garey et D.S. Johnson (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, San Fransisco

M. Grötschel, R. Borndörfer, et A. Löbel (2003). Duty scheduling in public transit. *Mathematics-Key Technologies for the Future*, Springer, Berlin, Germany. 653–674.

K. Haase, G. Desaulniers, et J. Desrosiers (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, *35*, 286–303.

K. Haase et C. Friberg (1999). An exact branch and cut algorithm for the vehicle and crew scheduling problem. *Computer-Aided Transit Scheduling*, Springer Verlag, Berlin, Germany. 63–80.

J. Halpern et J. Priess (1974). Shortest paths with time constraints on moving and parking. *Networks*, *4(3)*, 241–253.

G.Y. Handler et I. Zang (1980). A dual algorithm for the constrained shortest path problem. *Networks*, *10*, 293–309.

M. Heing (1986). The shortest path problem with two objective functions. *European Journal of Operational Research*, *25(2)*, 281–291.

I. Himmich, I. El Hallaoui, et F. Soumis (2018a). A multi-directional dynamic programming algorithm for the shortest path problem with resource constraints. *Cahier de GERAD, G-2018-05 (HEC Montreal).*

I. Himmich, H. Ben Amor, I. El Hallaoui, et F. Soumis (2018b). A primal adjacency-based algorithm for the shortest path problem with resource constraints. *Cahier de GERAD, G-2018-09 (HEC Montreal).*

I. Himmich, I. El Hallaoui, et F. Soumis (2018c). Primal column generation framework for vehicle and crew scheduling problems. *Cahier de GERAD, G-2018-74 (HEC Montreal).*

K. Holmberg et D. Yuan (2003). A multicommodity network-flow problem with side constraints on paths solved by column generation. *INFORMS Journal on Computing*, *15*, 42–57.

I. Ioachim, J. Desrosiers, Y. Dumas, M.M. Solomon, et D. Villeneuve (1995). A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, *29*, 63–78.

I. Ioachim, S. Gélinas, F. Soumis, et J. Desrosiers (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, *31(3)*, 193–204.

S. Irnich et G. Desaulniers (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M.M. Solomon (Editors), *Column Generation*, Springer, Boston. 33–65.

S. Irnich et D. Villeneuve (2006). The shortest path problem with resource constraints and k-cycle elimination for k $\geq$ 3. *INFORMS Journal on Computing*, *18(3)*, 391–406.

J.M. Jaffe (1984). Algorithms for finding paths with multiple constraints. *Networks*, *14(1)*, 95–116.

J.E Kelley Jr. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, *8(4)*, 703–712.

O. Laval, A. Nagih et S.Toulouse (2006). Rapport de recherche sur le problème du plus court chemin contraint. *Technical Report 2006-05, LIPN - CNRS UMR 7030, Université Paris 13*.

L. Lozano, D. Duque, et A.L. Medaglia (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, *50(1)*, 348–357.

L. Lozano et A.L. Medaglia (2013). On an exact method for the constrained shortest path problem. *Computers and Operations Research*, *40(1)*, 378–384.

K. Melhorn et M. Ziegelmann (2000). Resource constrained shortest paths. *Algorithms - European Symposium on Algorithms (ESA 2000), Part of the Lecture Notes in Computer Science book series (LNCS, volume 1879)*, Springer-Verlag, Berlin. 326–337.

A. Mingozzi, M.A. Boschetti, S. Ricciardelli, et L. Bianco (1999). A set partitioning approach to the crew scheduling problem. *Operations Research*, *47*, 873–888.

R. Muhandiramge et N. Boland (2009). Simultaneous solution of Lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem. *Networks*, *53*, 358–381.

A. Nagih et F. Soumis (2006). Nodal aggregation of resource constraints in shortest path problem. *European Journal of Operational Research*, *172(2)*, 500–514.

I. Patrikalakis et D. Xerocostas (1992). A new decomposition scheme of the urban public transport scheduling problem. In : M. Desrochers and J.M. Rousseau (Editors), *Proceedings of the Fifth Inter-national Workshop on Computer-aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems, vol. 386.*, Springer Verlag, Berlin. 407–425.

W.B. Powell et Z.-L. Chen (1998). A generalized threshold algorithm for the shortest path problem with time windows. In P. Pardalos and D. Du (Editors), *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 303–318. American Mathematical Society.

L.D.P. Pugliese et F. Guerriero (2013a). A reference point approach for the resource constrained shortest path problems. *Transportation Science*, *47(2)*, 247–265.

L.D.P. Pugliese et F. Guerriero (2013b). A survey of resource constrained shortest path problems : Exact solution approaches. *Networks*, *62(3)*, 183–200.

G. Righini et M. Salani (2006). Symmetry helps : Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, *3*, 255–273.

L. Santos, J. Coutinho-Rodrigues, et J.R. Current (2007). An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B : Methodological*, *41(7)*, 756–771.

D. Scott (1985). A large linear programming approach to the public transport scheduling and cost model. In J.M. Rousseau (Editor), *Computer Scheduling of Public Transport 2*, North-Holland, Amsterdam, 473–491.

D.J. White (1982). The set of efficient solutions for multiple objective shortest path problems. *Computers and Operations Research*, *9(2)*, 101–107.

W.E. Wilhelm, I. Arambula, et N.D. Choudhry (2006). Optimizing picking operations on dual-head placement machines. *IEEE Transactions on Automation Science and Engineering*, *3*, 1–15.

W.E. Wilhelm, N.D. Choudhry, et P. Damodaran (2007). A model to optimize placement operations on dual-head placement machines. *Discrete Optimization*, *4*, 232–256.

W.E. Wilhelm, P. Damodaran, et J. Li (2003). Prescribing the content and timing of product updates. *IIE Transactions*, *35*, 647–663.

G. Xue (2000). Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. *Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference*, 271–277.

J.Y. Yen (1971). Finding the K Shortest Loopless Paths in a Network. *Management Science*, *17(11)*, 661–786.

M. Zabarankin, S. Uryasev, et P. Pardalos (2001). Optimal risk path algorithms. *Cooperative Control and Optimization*, Murphey, R., Pardalos, P.M., eds., Kluwer, Dordrecht, the Netherlands. 271–303.

A. Zaghrouti, I. El Hallaoui, et F. Soumis (2014). Integral simplex using decomposition for the set partitioning problem. *Operations Research*, *62(2)*, 435–449.

F.B. Zhan (2000). A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. *Journal of Geographic Information and Decision Analysis*, *4(2)*, 1–11.

K. Ziarati, F. Soumis, J. Desrosiers, S. Gélinas, et A. Saintonge (1997). Locomotive assignment with heterogeneous consists at CN North America. *European Journal of Operational Research*, *97(2)*, 281–292.