Université de Montréal

Algorithmes pour le problème de repositionnement

par

Charles Bordenave

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

Docteur ès sciences (Ph.D.)

en informatique option recherche opérationnelle

Juin 2008

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Algorithmes pour le problème de repositionnement

présentée par:

Charles Bordenave

a été évaluée par un jury composé des personnes suivantes:

Jacques Ferland

(président-rapporteur)

Michel Gendreau

(directeur de recherche)

Gilbert Laporte

(co-directeur)

Bernard Gendron

(membre du jury)

Angel Corberan

(examinateur externe)

Thèse acceptée le:

Résumé

Cette thèse propose des modèles et des algorithmes pour un problème complexe d'optimisation dont les applications trouvent leur place dans des domaines tels que les transports ou la robotique. Le *Problème de Repositionnement* (PR) généralise plusieurs problèmes classiques de tournées de véhicule. Initialement chaque sommet d'un graphe contient au plus un objet de type donné qui représente son offre. Chaque sommet possède aussi une demande, en terme de type d'objet. Le type d'objet demandé peut possiblement être le même que celui qui est offert. Un véhicule de capacité unitaire est disponible pour repositionner les objets entre les sommets. Le PR consiste à déterminer la route la moins coûteuse pour repositionner les objets de telle sorte que toutes les demandes soient satisfaites. Il existe très peu de recherches traitant du cas général de ce problème, c'est-à-dire le cas où le graphe sous-jacent est complet. On distingue trois variantes du PR : la version non préemptive, la version préemptive et la version mixte. Dans la version préemptive, tous les objets sont *déposables*, c'est-à-dire que le véhicule est autorisé à temporairement les décharger sur des sommets intermédiaires avant de les transporter vers l'une de leurs destinations potentielles. Cette thèse présente pour la première fois des algorithmes exacts de coupes et branchements pour les trois variantes du PR définies sur un graphe complet. Elle propose également des heuristiques qui permettent de résoudre approximativement des instances de grande taille.

Mots clés. Problème de repositionnement, problème de tournée de véhicule, problème de transport, relations de préséance, algorithme de coupes et branchements, heuristique

Abstract

This thesis proposes models and algorithms for a complex optimization problem arising in transportation and robotics. The *Swapping Problem* (SP) is a generalization of some well known vehicle routing problems. Initially, each vertex of a graph contains at most one object type, which represents its supply. Each vertex also demands an object type. The demand may be equal to the supply. A unit capacity vehicle is available to carry the objects between the vertices. The SP consists of determining a minimum cost route that allows the vehicle to reposition the objects in such a way that all demands are satisfied. Very little research has been carried out on the general case of this problem, i.e., the case where the underlying graph is complete. One can distinguish between three different versions of the SP : the non-preemptive version, the preemptive version, and the mixed version. In the preemptive version, all objects are *droppable*, meaning that the vehicle is allowed to temporarily unload them at some intermediate vertices before carrying them to one of their potential destinations. This thesis presents, for the first time, exact branch-and-cut algorithms for the three versions of the SP defined on a complete graph. It also proposes heuristic allowing the solution of large scale instances.

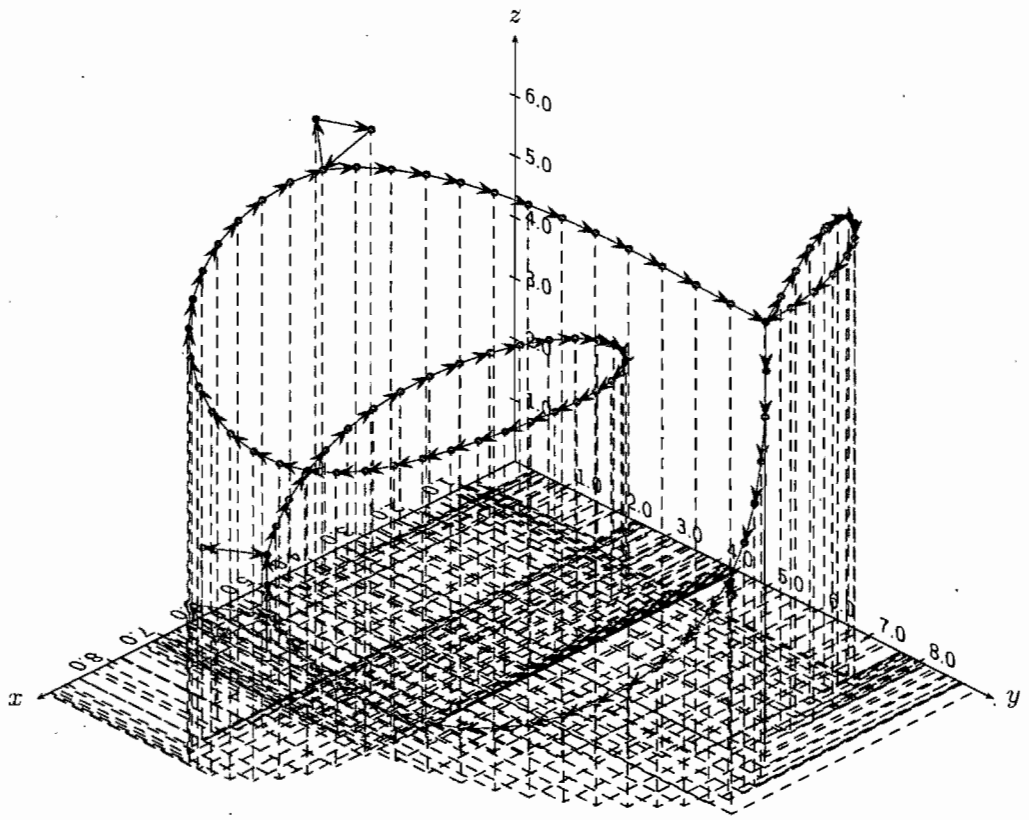
Keywords. Swapping Problem, vehicle routing, transportation problem, precedence relationships, branch-and-cut, heuristic

Table des matières

Liste des tableaux

Liste des figures

À Marie et Édouard



Représentation tridimensionnelle d'une solution optimale d'un problème de repositionnement

Remerciements

Bien qu'elle soit le résultat d'un travail personnel, une thèse ne peut être menée à terme sans un bon encadrement. Mes premiers remerciements seront naturellement adressés à mon directeur, le professeur Michel Gendreau, et à mon co-directeur, le professeur Gilbert Laporte, avec qui j'ai pris plaisir à travailler lors de ces cinq années de recherche. Du jour où le premier a accepté ma candidature, à il y a quelques semaines où le second corrigeait cette page, ils m'ont soutenu et encouragé. Je n'oublierai pas les reflexions et conseils avisés de Michel, ni les nombreuses discussions avec Gilbert sans lesquelles ce travail n'aurait pu prendre sa forme actuelle.

Je tiens également à exprimer ma gratitude à l'ensemble des membres du *Centre de recherche sur les transports*, devenu le *Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport*, pour leur accueil, leur soutien et leur gentillesse. Je pense entre autres à Serge Bisaillon et à François Guertin qui m'ont aidé à implémenter la plupart des algorithmes présentés dans cette thèse, à Julio Montecinos pour m'avoir épaulé dans les passages difficiles de la thèse et à Walter Rei pour les conversations fructueuses que l'on a pu avoir.

Finalement, comment pourrais-je ne pas remercier mes parents, Hélène et Max, qui m'ont supporté sans faille tout au long de la thèse depuis ma Charente-Maritime natale ? Ce projet n'aurait pu se réaliser sans eux, je leur suis infiniment reconnaissant.

Chapitre 1

Introduction

1.1 Préambule

Les problèmes de cueillette et livraison constituent une classe importante des problèmes de tournées de véhicule dans lesquels des objets (qui peuvent être des produits ou encore des personnes dans certaines applications) doivent être transportés d'une origine vers une destination. Ils sont souvent reliés à des problèmes réels rencontrés dans des domaines tels que les transports (transport en commun, transport de marchandises), la logistique ou encore la robotique. On les modélise en général à l'aide d'un graphe orienté $G = (V, A)$, où V représente l'ensemble des sommets et A l'ensemble des arcs sur lequel une matrice de coûts est définie. A chaque sommet de V est associée une offre ou une demande (ou les deux à la fois). Un véhicule (parfois plusieurs), commençant et terminant son tour à un sommet particulier appelé le *dépôt*, est disponible pour visiter l'ensemble des sommets en parcourant A. Le tour doit permettre au véhicule de satisfaire la demande de chaque sommet. Les contraintes régissant la façon dont les visites peuvent ou doivent se faire dépendent du problème traité. Le but est, dans la plupart des cas, de déterminer un tour de coût minimum.

Au sein de ces problèmes et des variantes qui leur sont rattachées, on peut distinguer trois principales familles : les problèmes de type *un-pour-un*, ceux de type *plusieurs-*

pour-plusieurs et enfin ceux de type *un-pour-plusieurs-pour-un* ([14]). À chaque famille correspondent des contraintes spécifiques : contraintes sur le nombre de visites autorisées pour un même sommet, contraintes sur l'offre et la demande des sommets, contraintes sur le nombre de véhicules disponibles, contraintes sur la capacité du ou des véhicules, contraintes sur des fenêtres de temps, etc. Le chapitre 2 présente une description détaillée de certains d'entre eux. Ces problèmes ont été largement étudiés au cours des trente dernières années et différentes approches de résolution ont été proposées qu'elles soient de nature heuristique ou exacte.

Le sujet de cette thèse est le *Problème de Repositionnement* (PR), que l'on appelle en anglais le *Swapping Problem*. Le PR correspond a un problème de type plusieurs-pour-plusieurs, dans le sens où chaque objet peut être potentiellement transporté vers plusieurs destinations et réciproquement la livraison peut provenir de différentes origines. Dans le PR on dispose d'un unique véhicule de capacité unitaire (i.e., il ne peut transporter plus d'un objet à la fois), qui débute et termine son tour au sommet dépôt. Une instance du PR est caractérisée d'une part par la matrice de coûts et d'autre part par la position initiale et finale de chaque type d'objet. Pour faire une analogie avec les autres problèmes de cueillette et livraison, on peut considérer que dans le PR il existe une offre et une demande unitaire sur chaque sommet qui s'expriment en terme de type d'objet. L'objectif est de déterminer un tour de coût minimum tel que toutes les demandes soient satisfaites, c'est-à-dire déterminer un tour qui permette au véhicule de repositionner à moindre coût chaque objet sur un sommet adéquat.

Les problèmes de type plusieurs-pour-plusieurs ont été relativement peu étudiés comparativement aux problèmes de type un-pour-un qui constituent la famille de problèmes sur laquelle le plus d'effort a été concentré, sans doute en raison de leurs nombreuses implications dans la résolution de problèmes rencontrés fréquemment dans l'industrie. Dans les problèmes de type plusieurs-pour-plusieurs, contrairement à beaucoup de problèmes de cueillette et livraison, la destination de chaque objet pris individuellement n'est pas connue a priori. Les objets étant classés par type et deux objets d'un même type étant identiques, chaque objet peut être déplacé vers un sommet qui demande ce type d'objet mais qui ne demande pas forcément cet objet en particulier.

Ces problèmes peuvent être vus comme une généralisation de certains problèmes un-pour-un. Il faut implicitement résoudre un problème d'affectation pour déterminer le déplacement des objets et un *Problème du Voyageur de Commerce* (PVC) dans lequel chaque sommet peut être visité plusieurs fois. Cette variante du PVC est connue sous le nom de la *Relaxation Graphique du Problème du Voyageur de Commerce* ([22]).

Il existe plusieurs versions au sein du PR. On peut d'une part envisager différentes structures de graphe (graphe général, arbre, cercle, ligne) et d'autre part considérer des objets qui peuvent être ou non déposés à des sommets intermédiaires avant d'être acheminés vers leur destination finale, on parle alors de *préemption.* Le cas *mixte* existe également (la préemption n'est permise que pour un sous-ensemble des types d'objet). Certaines versions peuvent se résoudre par des algorithmes polynomiaux, comme par exemple le cas où les sommets sont positionnés le long d'une ligne ([2]) ou sur un arbre lorsque le nombre de types d'objet est égal à 2 ([3]).

Dans le PR le véhicule est autorisé à voyager à vide entre deux sommets, i.e., à ne transporter aucun objet le long d'un arc. Il est commode de considérer qu'il transporte alors l'*objet nul*, noté 0, le long d'un tel arc. On appelle ce type d'arc un *parcours à vide* (*deadheading* en anglais). Un tour doit impérativement débuter et terminer au sommet dépot. Le dépot est un sommet choisi arbitrairement parmi l'ensemble des sommets et possède également une offre et une demande. On peut noter que le choix du sommet dépôt n'a aucune influence sur la solution lorsque la préemption n'est pas permise alors qu'il joue un rôle important dans le cas avec préemption. En effet, le dépôt initialise les *relations de préséance* qui stipulent logiquement qu'un objet ne peut être rechargé à un sommet avant d'y avoir été déposé (voir chapitre 4).

Le PR a été introduit par Anily et Hassin [4] qui ont montré que le PR défini sur un graphe complet est un problème NP-difficile par réduction au PVC. Ils ont exhibé certaines propriétés de structure des solutions optimales et ont proposé un algorithme 2.5-approché pour résoudre le PR dans le cas général. C'est à l'heure actuelle, et à notre connaissance, la seule étude traitant du cas général du PR.

1.2 Objectifs

Cette thèse a pour objectif principal de résoudre de façon exacte le PR à l'aide d'un algorithme de coupes et branchements (*branch-and-cut algorithm* en anglais). On se place dans le cadre général d'un graphe complet pour la structure sous-jacente, c'est-à-dire que le véhicule peut accéder à n'importe quel sommet à partir de n'importe quel autre en empruntant un seul arc. On souhaite étudier trois versions de ce problème : le cas sans préemption, le cas avec préemption et enfin la version mixte.

Il n'existe pas à notre connaissance de telle approche pour résoudre ce problème. Les précédentes études relatives au PR ont considéré des structures de graphe simplifiées (comme la ligne ou l'arbre) ou proposé des méthodes de résolutions approximatives. L'objectif est donc d'une part de proposer un certain nombre de formulations mathématiques pour modéliser les différentes versions étudiées et d'autre part d'élaborer pour chacune d'elles un algorithme exact pour les résoudre.

L'objectif secondaire est de proposer des méthodes heuristiques pour résoudre approximativement la version mixte du PR. Celles-ci doivent permettre de résoudre des instances de grande taille en un temps de calcul plutôt court, tout en produisant des solutions de bonne qualité (i.e., des solutions dont l'écart par rapport à la solution optimale est faible).

1.3 Plan de la thèse

Cette thèse comprend six chapitres qui peuvent être lus indépendamment l'un de l'autre. Les chapitres 3, 4, et 5 sont en anglais et ont été soumis pour publication.

Le chapitre 2 porte sur la littérature associée au PR. On y présente d'abord les travaux sur le PR lui-même, puis on dérive vers des travaux sur des problèmes connexes comme le *Problème de la Grue* ou le *Problème du Voyageur de Commerce Biparti.*

Le chapitre 3 étudie la version préemptive du PR. Un modèle mathématique et un

algorithme de coupes et branchements sont proposés. Cet algorithme a été appliqué avec succès sur des instances géométriques aléatoires contenant jusqu'à 200 sommets et huit types d'objet. L'article correspondant à ce chapitre a été soumis à *Naval Research Logistics.*

Le chapitre 4 traite le cas où la préemption est permise pour chaque type d'objet. Le fait d'autoriser le véhicule à déposer temporairement un objet sur un sommet complique considérablement la tâche. Certaines propriétés de structure des solutions optimales sont exhibées, puis un modèle et un algorithme de coupes et branchements sont présentés. Pour cette version des problèmes contenant jusqu'à 100 sommets et huit types d'objet sont résolus optimalement. Il est montré à la fin de ce chapitre comment l'algorithme peut être modifié pour résoudre optimalement le cas mixte du PR. L'article correspondant à ce chapitre a été soumis à *Mathematical Programming.*

Le chapitre 5 est consacré à plusieurs heuristiques que l'on a développées pour la version mixte du PR. Ces heuristiques ont été testées sur des instances contenant jusqu'à 10,000 sommets. Les temps de calcul sont acceptables et l'écart relatif moyen par rapport à la borne inférieure est de l'ordre de 1%. L'article correspondant à ce chapitre a été soumis à *Computers & Operations Research.*

Le chapitre 6 rassemble les grandes lignes de cette thèse en résumant les différents éléments apportés. Certaines améliorations possibles sont proposées et quelques idées de recherche pour de futurs travaux sont émises.

1.4 Contribution

Le contenu de cette thèse a été principalement élaboré et rédigé par son auteur. Il est par conséquent à l'origine de l'ensemble des résultats théoriques et numériques présentés dans les chapitres 3, 4 et 5.

Chapitre 2

Revue de la littérature

2.1 Le Problème de Repositionnement

On décrit dans cette section les principaux travaux relatifs au *Problème de Repositionnement*. On commence par revoir la littérature consacrée au PR dans le cas où la structure sous-jacente est le graphe complet et on analyse ensuite le cas particulier de la ligne puis de l'arbre.

2.1.1 Le cas général

Le PR a été introduit par Anily et Hassin [4]. Les auteurs définissent le problème de la manière suivante (traduction libre). « Chaque sommet d'un graphe contient initialement un objet de type connu. Un état final spécifiant les types d'objet désirés sur chaque sommet est également donné. Un unique véhicule de capacité unitaire est disponible pour transporter les objets à travers les sommets. Résoudre le *Problème de Repositionnement* consiste à déterminer la route la plus courte qui permette au véhicule de repositionner chaque objet de telle sorte que l'état final soit atteint. »

Le véhicule doit débuter et terminer sa route au sommet dépôt et il est autorisé à emprunter des parcours à vide (arcs le long desquels il ne transporte pas d'objet). La

distance entre les sommets est supposée respecter l'inégalité triangulaire. Les auteurs considèrent le cas général d'un graphe complet et définissent l'ensemble des types d'objet O comme l'union des objets déposables temporairement (O_d) et ceux devant être directement acheminés (O_n), i.e., $O = O_n \cup O_d$. Ils autorisent l'offre et la demande à être identiques pour un sommet donné (sommet qui se sert lui-même) et l'offre totale est égale à la demande totale pour s'assurer de la réalisabilité.

La figure 2.1 illustre le problème en présentant une petite instance du PR avec cinq sommets et cinq types d'objet. La préemption est ici autorisée pour tous les sommets (i.e., $O = O_d$). À chaque sommet est associée une paire d'objets correspondant à l'état initial et final. Le sommet symbolisé par un carré représente le dépôt. La figure 2.2 représente une solution optimale mais non unique pour cette instance. Lorsqu'un objet est transporté le long d'un arc, le type de cet objet est noté sur l'arc. On remarque que le véhicule dépose temporairement l'objet de type 5 sur le sommet situé dans le coin inférieur droit avant de le transporter sur le sommet situé dans le coin supérieur droit. Aucun parcours à vide n'est utilisé ici.

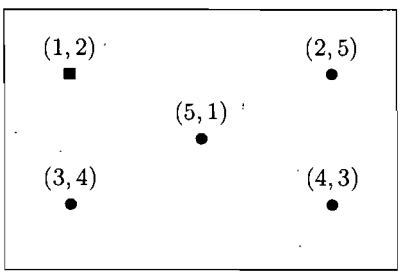


FIG. 2.1 – Instance du PR ($O = O_d$)

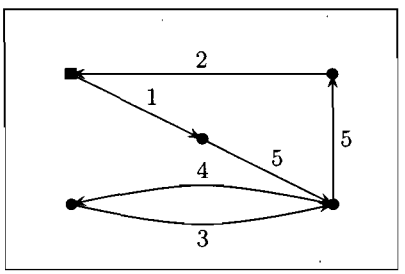


FIG. 2.2 – Une solution optimale

On peut voir assez facilement que ce problème est NP-difficile en considérant une instance du *Problème du Voyageur de Commerce* (PVC) dans laquelle on attribue à chaque sommet une offre et une demande. On duplique ensuite chacun des sommets en plaçant sa copie à une distance nulle et on permute l'offre et la demande (i.e., l'un satisfait la demande de l'autre et réciproquement, voir figure 2.4). Il est clair que résoudre l'instance du PR sur ce nouveau graphe résout également l'instance du PVC sur le graphe d'origine (figure 2.3).

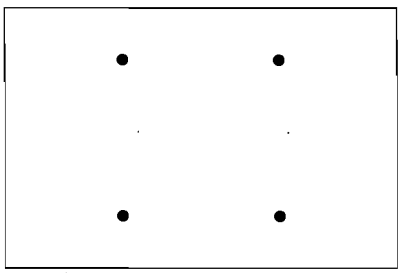


FIG. 2.3 – Instance du PVC

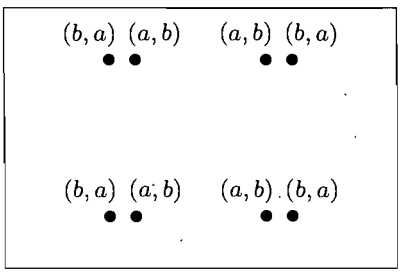


FIG. 2.4 – Transformation en PR

Parmi l'ensemble des solutions qui minimisent la fonction objectif, celles qui comportent le moins d'arcs sont dites *optimales*. Les auteurs démontrent plusieurs propriétés importantes des solutions optimales comme le fait que le nombre de visites d'un sommet pour le cas mixte du PR est borné par trois, ou comme le fait que les arcs incidents à un sommet respectent un certain ordre de passage. Ils prouvent que si le véhicule dépose temporairement un objet sur un sommet, alors il le fera lors de la première visite du sommet et le rechargement de cet objet correspondra à la sortie de la dernière visite. De la même manière un parcours à vide ne peut être utilisé dans une solution optimale que lors de la première entrée dans le sommet et au cours de la dernière sortie.

Les auteurs montrent que lorsque $O = O_d$, le nombre maximum de visites pour un sommet se réduit à deux. En d'autres termes, dans cette version où la préemption est permise pour chaque objet, aucun parcours à vide additionnel n'appartient à une solution optimale. Cela signifie que lorsqu'il n'existe pas d'objet nul dans l'instance (i.e., lorsque tous les sommets possèdent initialement une offre et une demande), il existe une solution optimale dans laquelle le véhicule transporte en permanence un objet. Leur démonstration se base sur le fait que si dans une solution optimale l'ordre de passage des arcs ou le degré des sommets étaient différents de celui qu'ils précisent, alors une nouvelle route ayant un coût inférieur ou égal et comportant moins d'arcs pourrait être construite en permutant certains arcs et en remplaçant les paires d'arcs consécutifs transportant un même type d'objet par un arc unique. On peut déduire de ces observations que dans une solution optimale le véhicule ne déposera jamais temporairement un objet sur un sommet qui offre ou demande déjà un objet de même type.

Les auteurs proposent un algorithme 2.5-approché pour résoudre le cas général du PR en adaptant une heuristique connue pour le PVC qui consiste à agglomérer des cycles. La procédure considère d'abord une relaxation du problème en résolvant un problème d'affectation de coût minimum pour chaque type d'objet (reliant l'offre à la demande et la demande à l'offre). Un ensemble de cycles est ainsi obtenu. La deuxième étape consiste à choisir arbitrairement un représentant au sein de chaque cycle et de résoudre approximativement un PVC sur ces sommets. Finalement la solution réalisable est construite en combinant les cycles obtenus lors de la première phase et la solution du PVC. Parfois des raccourcis peuvent être ensuite effectués à l'aide de déchargement temporaire d'objet ou lorsque deux arcs consécutifs transportent le même type d'objet.

Les figures 2.5–2.8 illustrent brièvement leur algorithme. Le facteur d'approximation 2.5 provient du fait que leur heuristique résout un problème d'affectation de coût minimum dont la valeur de l'objectif est inférieure ou égale à l'optimum et un PVC par un algorithme approché comme celui de Christofides ([18]) dont la valeur de la solution est inférieure ou égale à 1.5 fois l'optimum.

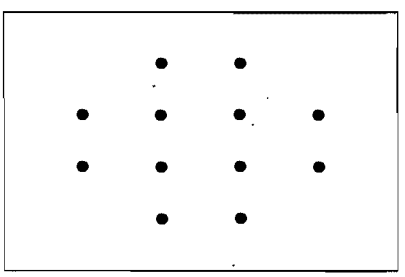


FIG. 2.5 – Instance

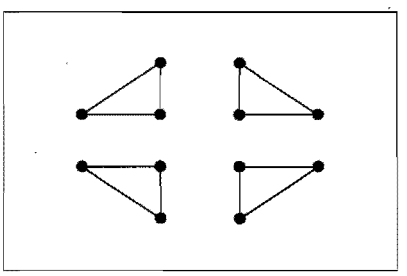


FIG. 2.6 – Affectation minimum

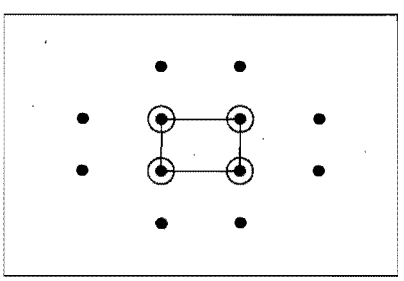


FIG. 2.7 – PVC sur les représentants

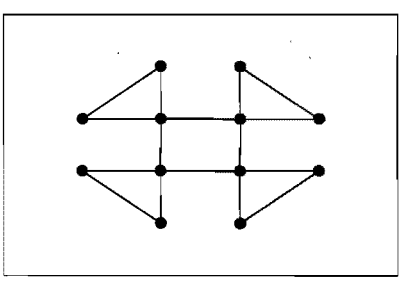


FIG. 2.8 – Solution réalisable

Les auteurs proposent un deuxième algorithme qui commence par résoudre le même

problème d'affectation que dans le premier algorithme mais qui calcule ensuite un arbre couvrant sur des représentants de chaque cycle ($u \in C_i$ et $v \in C_j$ sont choisis en tant que représentants de C_i et C_j, deux cycles obtenus lors de la première étape de l'algorithme, si c_{uv} minimise la distance entre C_i et C_j) et le combine avec la solution d'un couplage parfait sur les sommets de degré impair. Cette seconde méthode n'améliore pas le facteur d'approximation 2.5 du premier algorithme.

2.1.2 Le cas de la ligne

Anily, Gendreau et Laporte [2] analysent le cas particulier du PR où les sommets du graphe sont situés sur une ligne. Ils montrent que cette version peut se résoudre en temps polynomial en proposant un algorithme exact d'ordre $O(n^2)$.

Le fait de réduire d'une dimension la structure géométrique sous-jacente (i.e., de passer d'un plan à une droite) implique la propriété intéressante suivante : si l'on considère l'ensemble des chemins transportant l'objet de type k dans une solution (i.e., la séquence d'arcs le long desquels le véhicule déplace k), on peut montrer qu'il existe une solution optimale dans laquelle ces chemins ne se croisent pas et ne sont pas imbriqués (dans une direction opposée). Comme le montre la figure 2.10, on peut toujours transformer une solution qui contiendrait ce type de chemins par une solution tout aussi bonne qui n'en contiendrait pas. La route reste identique et seule l'affectation des objets sur les arcs change. Une paire de parcours à vide est utilisée pour la jonction.

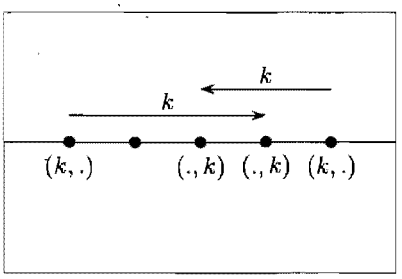


FIG. 2.9 – Chemins croisés

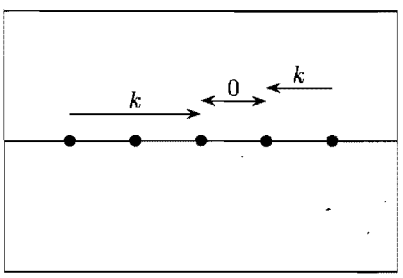


FIG. 2.10 – Alternative

Les auteurs montrent également que pour ce sous-ensemble de solutions optimales, la longueur des chemins transportant un objet de type k est identique au sein de

chaque solution et que les chemins de transport associés à un type d'objet induisent une partition particulière des sommets qu'ils appellent *partition minimum consécutive équilibrée*. Leur algorithme fonctionne en deux étapes. Tout d'abord il construit pour chaque type d'objet un ensemble d'arcs qui repositionne les objets à un coût minimum et de telle sorte que les sommets ont le même nombre d'arcs entrants et sortants. Cette phase de leur algorithme correspond à une technique d'augmentation de graphe. Le graphe obtenu n'étant pas nécessairement connexe, la deuxième étape consiste à identifier les différentes composantes connexes et à rechercher un arbre couvrant de coût minimum sur les représentants des composantes (les représentants sont choisis de telle sorte qu'ils minimisent la distance inter-composantes). Les arêtes de cet arbre sont enfin dédoublées et transformées en arcs dont la direction est opposée afin de respecter la parité des degrés et d'avoir un graphe eulérien.

La méthode utilisée ressemble d'une certaine façon à l'heuristique proposée dans [4] pour le cas du graphe complet, dans le sens où cette procédure cherche à minimiser séparément (pour chaque type d'objet) la longueur totale des chemins de transport et à relier ensuite à moindre coût les différentes composantes entre elles à l'aide de parcours à vide ou éventuellement de déchargement temporaire. En revanche, comme le montrent les auteurs, cette politique est optimale dans le cas de la ligne. L'ordre de complexité $O(n^2)$ de leur algorithme correspond à la complexité du calcul de l'arbre couvrant (la première phase pouvant se réaliser en temps $O(n)$).

2.1.3 Le cas de l'arbre

Anily et al. [3] examinent le cas où les sommets du graphe sont disposés sur une structure d'arbre (figure 2.11). Ils prouvent que cette version du PR est NP-difficile par réduction au *Problème de l'Arbre de Steiner* dans un graphe biparti.

À l'aide du même procédé d'augmentation de graphe que celui qu'ils ont utilisé pour le cas de la ligne, ils développent un algorithme qui génère des solutions réalisables dont le coût est inférieur ou égal à 1.5 fois la valeur d'une solution optimale. Ils montrent que cette borne peut être atteinte pour certaines instances (borne serrée).

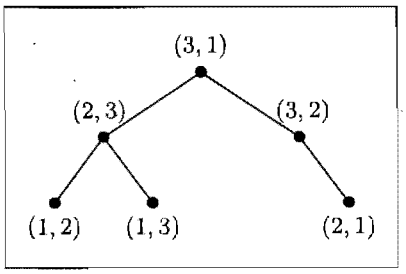


FIG. 2.11 – PR sur un arbre

L'algorithme commence par construire sur les sommets de l'arbre un premier graphe dans lequel chaque composante est équilibrée en terme d'offre et de demande par type, et pour tous les sommets le nombre d'arcs entrants et sortants est identique. Le coût total associé à ces arcs représente une borne inférieure sur la valeur d'une solution optimale. La structure obtenue, un multi-graphe (multi-types), n'étant pas nécessairement connexe, la deuxième phase vise à connecter itérativement les différentes composantes. Les auteurs montrent que le coût d'une telle augmentation est au plus égal à la moitié du coût des arcs déjà construits. Ceci mène à un algorithme 1.5-approché.

Pour le cas particulier où il n'existe que deux types d'objet à repositionner (incluant l'objet nul), il est montré que le graphe obtenu après la première étape est fortement connexe et que ses arcs constituent une solution optimale. Ce cas particulier peut donc se résoudre en temps polynomial.

2.2 Quelques problèmes voisins

On décrit dans cette section quelques problèmes reliés de façon plus ou moins lointaine au PR. Certains d'entre eux sont des cas particuliers du PR avec par exemple une restriction sur le nombre de types d'objet considérés ou sur le nombre d'objets de chaque type. Ces problèmes ont été historiquement introduits avant le PR.

2.2.1 Le Problème du Voyageur de Commerce Biparti

Le *Problème du Voyageur de Commerce Biparti* (PVCB) est une variante du classique PVC. L'objectif est de déterminer dans un graphe $G = (V = V^+ \cup V^-, E)$ un cycle hamiltonien de longueur minimum traversant tour à tour un sommet de V^+ et un sommet de V^- (figures 2.12 et 2.13). Ce problème est un cas particulier du PR dans lequel il n'existe que deux types d'objet (incluant l'objet nul). On le retrouve principalement dans des problèmes liés à la robotique.

Chalasani et Motwani [17] proposent un algorithme 2-approché pour résoudre le PVCB . Le facteur 2 correspond à deux fois le coût d'un arbre couvrant biparti minimum dans lequel les sommets de V^+ sont de degré au plus 2 (le coût de l'arbre couvrant représentant une borne inférieure sur le coût optimal). Les auteurs montrent qu'effectuer une traversée en profondeur de l'arbre en créant des raccourcis lorsque l'on arrive aux feuilles conduit à une solution réalisable qui n'est pas plus longue que le double d'une solution optimale. Ils prouvent que la recherche d'un tel arbre peut se faire en temps polynomial en interprétant le problème comme un cas particulier de problème d'intersection de deux matroïdes ([25, 26]). La première matroïde est la matroïde des forêts biparties, la seconde est la matroïde représentant les sous-graphes bipartis dont les sommets de V^+ sont de degré au plus 2.

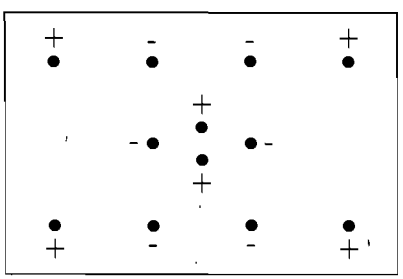


FIG. 2.12 – Instance du PVCB

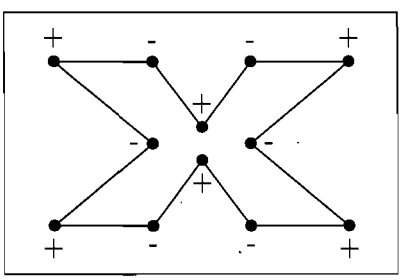


FIG. 2.13 – Solution réalisable du PVCB

Le facteur d'approximation est meilleur que le facteur d'approximation de l'algorithme proposé par [4] pour le PR. En revanche, cet algorithme ne produit pas nécessairement de meilleurs résultats en pratique. Dans [13] les auteurs proposent une étude comparative de différents algorithmes pour résoudre le PVCB. Ils comparent les

solutions générées par l'algorithme proposé par [17] (facteur d'approximation 2), celui proposé par [4] (facteur d'approximation 2.5) et un autre algorithme qui consiste à déterminer une couverture par cycles de coût minimum (cycles bipartis) et à relier ces cycles à l'aide d'un arbre couvrant de coût minimum (facteur d'approximation 3). Leurs tests numériques sont effectués sur un lot de plusieurs centaines d'instances aléatoires du PVCB comprenant entre 40 et 160 sommets regroupés en quatres catégories. L'algorithme présenté dans [17], bien qu'ayant la borne de pire cas la plus petite, se révèle fournir les solutions les plus pauvres dans chacune des catégories.

Ghiani et al. [31] ont considéré une généralisation du PVCB dans laquelle la cardinalité des ensembles V^+ et V^- peut être différente et le nombre de sommets de V^- entre deux sommets consécutifs de V^+ n'excède pas une certaine valeur. Ils appellent ce problème le *Problème du Voyageur de Commerce Noir et Blanc* ou *Black and White Traveling Salesman Problem* en anglais (les sommets de V^+ pouvant être colorés en noir et ceux de V^- en blanc). Ils présentent un algorithme de coupes et branchements pour ce problème NP-difficile.

2.2.2 Le Problème de la Grue

Le *Problème de la Grue* (PG), appelé *Stacker Crane Problem* en anglais, est défini sur un graphe mixte $G = (V, E \cup A)$, où V représente l'ensemble des sommets, E celui des arêtes et A celui des arcs. Un coût est associé à chaque arête et à chaque arc. L'objectif consiste à déterminer un circuit de coût minimum traversant les arcs de A au moins une fois (voir figures 2.14 et 2.15, par souci de clarté seul A a été dessiné sur la figure 2.14).

Le PG correspond à un cas particulier du PR dans lequel il n'existe qu'un seul exemplaire d'objet pour chacun des types, impliquant que les objets ont une origine et destination uniques. En d'autres termes, une partie de la solution est donnée et le but est de la compléter pour former un circuit optimal. Tout comme le PR, ce problème est NP-difficile dans le cas général (par réduction au PVC).

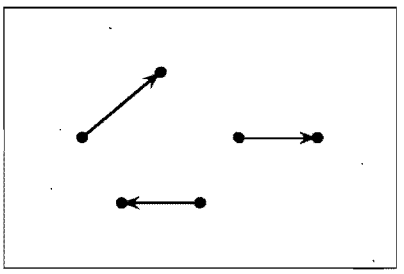


FIG. 2.14 – Instance du PG

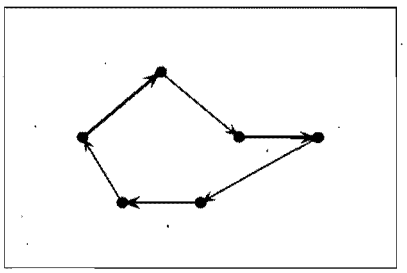


FIG. 2.15 – Solution réalisable du PG

Le cas général

Frederickson et al. [30] étudient le PG sur un graphe complet et proposent une heuristique de facteur d'approximation 1.8, qui fonctionne à la fois pour la version préemptive et la version non préemptive. Leur heuristique est constituée de deux algorithmes, le premier convenant mieux aux instances dont le coût des arcs imposés est important par rapport au coût moyen, le deuxième étant plus efficace sur des instances dont le coût des arcs imposés est relativement faible. Les deux algorithmes sont illustrés à partir de l'instance de la figure 2.14.

Leur premier algorithme commence par déterminer un couplage de coût minimum entre les destinations et les origines (en orientant les arcs des destinations vers les origines). Un ensemble de circuits disjoints est obtenu après cette étape. L'algorithme cherche alors un arbre couvrant de coût minimum sur les représentants de chaque circuit. Les arêtes de l'arbre sont dupliquées et transfomées en arcs ayant une direction opposée. Une solution réalisable du PG peut finalement s'obtenir en parcourant l'ensemble des arcs construits et une éventuelle optimisation de la solution peut se faire en ayant recours à des raccourcis (figures 2.16–2.19).

Leur deuxième approche vise à d'abord contracter les arcs de A (i.e., chaque paire origine-destination est contractée en un sommet), puis à déterminer un arbre couvrant de coût minimum sur ces nouveaux sommets. Un couplage de coût minimum est ensuite déterminé sur les sommets de degré impair. L'algorithme considère à nouveau les sommets d'origine et ajoute à tout arc dont les extrémités sont de degré impair un arc dans la direction opposée. La solution réalisable est finalement obtenue en combinant

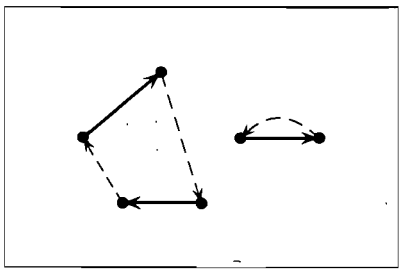


FIG. 2.16 – Couplage

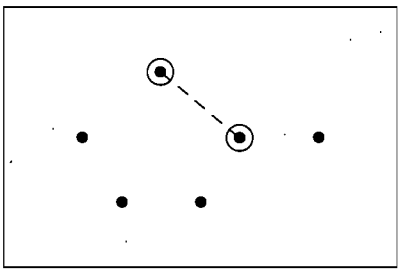


FIG. 2.17 – Arbre couvrant sur les représentants

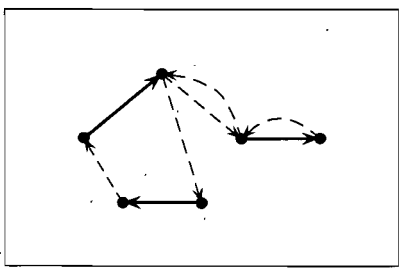


FIG. 2.18 – Formation de la solution

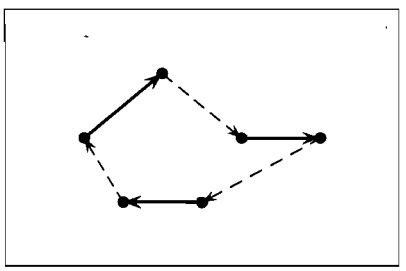


FIG. 2.19 – Raccourcis

les arcs construits et en effectuant des raccourcis le cas échéant (figures 2.20–2.23). Il est a priori impossible de déterminer lequel de ces deux algorithmes fournira la meilleure solution, les auteurs proposent par conséquent de les appliquer successivement et de ne retenir que la solution la plus intéressante.

Le cas de la ligne et du cercle

Au cours d'une étude sur des problèmes d'optimisation liés au mouvement du bras d'un robot, Atallah et Kosaraju [10] analysent les cas particuliers du PG où les sommets sont situés sur une ligne (figure 2.24) et sur un cercle (figure 2.26). Les auteurs montrent que ces deux problèmes peuvent se résoudre en temps polynomial en présentant un algorithme exact pour chacun. Ils considèrent la version prémptive et la version non préemptive.

Leurs algorithmes consistent à construire un graphe connexe et eulérien à partir des arcs imposés en utilisant une procédure d'augmentation de graphe. L'augmentation est

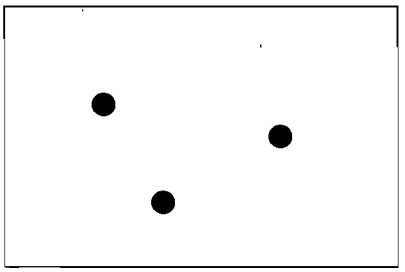


FIG. 2.20 – Contraction

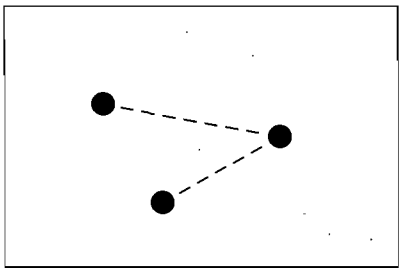


FIG. 2.21 – Arbre couvrant

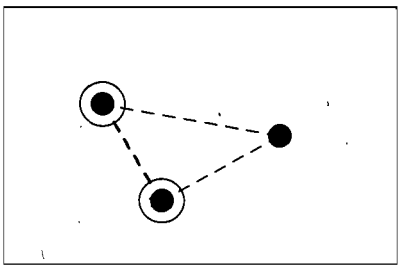


FIG. 2.22 – Couplage sur les sommets impairs

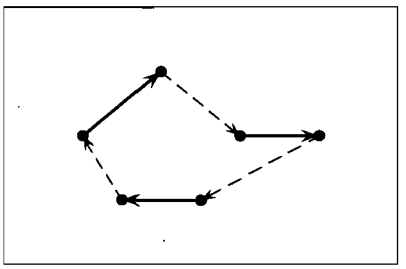


FIG. 2.23 – Retour aux sommets d'origine

réalisée de telle sorte que le graphe est équilibré en terme de flot entrant et sortant à chaque sommet, elle est de plus de coût minimum.

Pour le cas de la ligne, l'algorithme proposé est d'ordre $O(m + n\,\alpha(n))$ pour la version non préemptive, où m représente le nombre de types d'objet, n le nombre de sommets et α l'inverse d'une fonction d'Ackermann. Lorsque la préemption est autorisée leur algorithme est d'ordre $O(m + n)$.

Le cas du cercle est plus complexe techniquement que le cas de la ligne car à chaque arc $(i, j) \in A$ correspondent deux voies possibles pour rejoindre j à partir de i : le plus court chemin pour aller de i à j (voie *mineure*) ou son complémentaire en empruntant la direction opposée (voie *majeure*). Les auteurs montrent qu'il est parfois préférable d'emprunter une voie majeure et qu'une solution optimale n'en utilisera jamais plus d'une. Si l'on considère l'instance représentée sur la figure 2.26, la solution optimale utilise un déchargement temporaire le long d'une voie majeure (figure 2.27). Les auteurs proposent un algorithme d'ordre $O(m + n \log n)$ pour la version non préemptive et

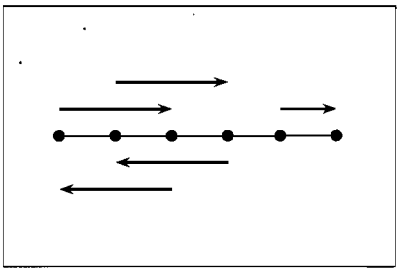


FIG. 2.24 – PG sur une ligne ($O = O_n$)

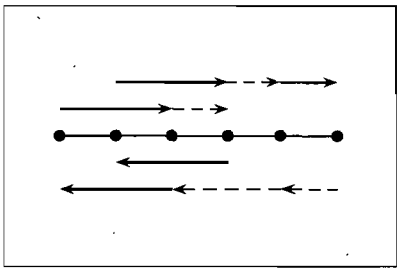


FIG. 2.25 – Solution optimale

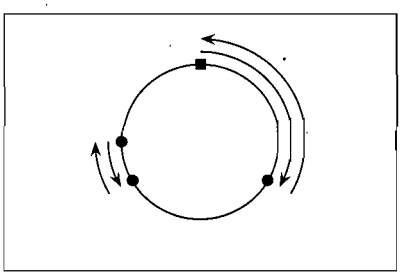


FIG. 2.26 – PG sur un cercle ($O = O_d$)

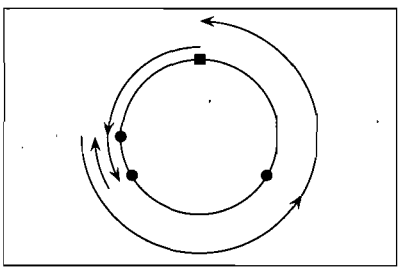


FIG. 2.27 – Solution optimale

d'ordre $O(m+n)$ pour la version préemptive.

Le cas de l'arbre

Frederickson et Guan [28] étudient la version préemptive du PG sur une structure d'arbre. Ils prouvent que ce cas particulier du PG peut se résoudre en temps polynomial et proposent deux algorithmes de complexité $O(m + qn)$ et $O(m + n \log n)$, où m représente le nombre de types d'objet, n le nombre de sommets de l'arbre et $q \leq \min\{m, n\}$ le nombre de composantes connexes non triviales dans un graphe orienté particulier. Leur approche est basée sur une augmentation de graphe qui conserve une structure équilibrée (i.e., pour chaque sommet le nombre d'arcs entrants et sortants sont égaux). Leur algorithme relie ensuite les composantes à l'aide d'un arbre couvrant de coût minimum.

Frederickson et Guan [29] analysent la version non préemptive du PG sur un arbre. Ils montrent que ce problème est NP-difficile et proposent deux heuristiques pour cette version : l'une de facteur d'approximation 1.5, basée sur le calcul d'un arbre de Steiner,

l'autre de facteur 1.25. La première s'exécute en temps $O(m+n)$.

2.2.3 Le Problème du Voyageur de Commerce avec Cueillette et Livraison d'un Produit

Le *Problème du Voyageur de Commerce avec Cueillette et Livraison d'un Produit* (m-PVCCLP lorsque m produits sont considérés), appelé *One-Commodity Pickup and Delivery Traveling Salesman Problem* en anglais, consiste en un PVC dans lequel le véhicule, débutant et terminant son tour au sommet dépôt, doit collecter et distribuer un unique produit à travers un graphe. Pour chaque offre (demande) du produit considéré il existe plusieurs destinations (sources) possibles. Les sommets différents du dépôt sont partitionnés en deux sous-ensembles : ceux qui offrent le produit et ceux qui le demandent. Contrairement au PR, dans le 1-PVCCLP les sommets possèdent une offre ou une demande (mais pas les deux à la fois), qui peut être n'importe quelle quantité entière du produit considéré (positive pour une offre, négative pour une demande, voir figures 2.28 et 2.29). Une autre différence avec le PR est que dans le 1-PVCCLP le véhicule possède une capacité $Q \geq 1$ lui permettant de transporter plusieurs unités du produit à la fois. L'objectif du 1-PVCCLP est de déterminer un circuit hamiltonien de coût minimum tel que le véhicule puisse satisfaire toutes les demandes.

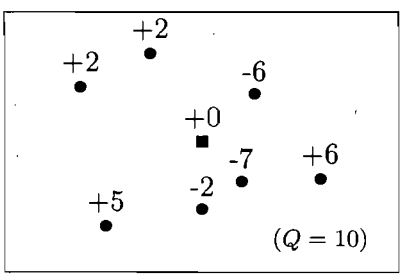


FIG. 2.28 – Instance du 1-PVCCLP

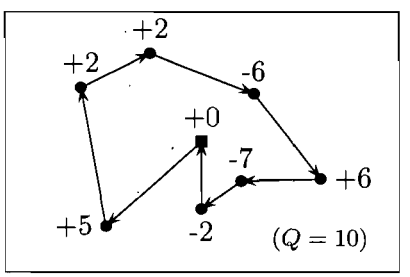


FIG. 2.29 – Solution réalisable

Ce problème est NP-difficile et la détermination même de la réalisabilité est NP-difficile. Il a été principalement étudié par Hernández-Pérez et Salazar González ([35, 36]) qui ont développé un algorithme de coupes et branchements pour le résoudre. Leur modèle de base est un modèle 0-1 incluant les contraintes classiques du PVC.

Au cours de la phase de séparation, en plus des usuelles contraintes de connectivité, ils ajoutent des contraintes liées à la capacité du véhicule (coupes de Benders) ainsi que des inégalités de cliques. Les auteurs parviennent à résoudre optimalement des instances du 1-PVCCLP contenant jusqu'à 50 sommets. Il apparaît que plus la capacité du véhicule est petite par rapport à l'offre et la demande moyenne des sommets, plus le problème devient difficile à résoudre. Les même auteurs proposent dans [36] une amélioration de cet algorithme en utilisant de nouvelles inégalités valides (inégalités multi-étoiles), [40] et [42]). Ils parviennent à résoudre des instances contenant jusqu'à 100 sommets.

2.2.4 Le Problème du Chemin Hamiltonien Asymétrique avec Contraintes de Préséance

Le *Problème du Chemin Hamiltonien Asymétrique avec Contraintes de Préséance* (PCHACP), appelé en anglais *The Asymmetric Hamiltonian Path Problem with Precedence Constraints* ou encore *The Sequential Ordering Problem*, se définit de la manière suivante. Un graphe complet orienté $G = (V, A)$ sur n sommets et une matrice de coûts (c_{ij}) définie sur A sont donnés. Un graphe auxiliaire $P = (V, R)$ spécifie des relations de préséance entre les sommets de V, i.e., un arc $(i, j) \in R$ indique que le sommet i doit précéder le sommet j. Le but est de déterminer dans G un chemin hamiltonien de coût minimum qui ne viole aucune contrainte de préséance.

Le PCHACP a été introduit par Escudero [27] qui propose une méthode heuristique pour le résoudre. Dans [7], [8], et [6], des algorithmes sophistiqués de coupes et branchements sont présentés pouvant résoudre optimalement des instances contenant plus d'une centaine de sommets. On mentionne le PCHACP dans cette revue de la littérature car dans la version préemptive du PR, le fait de déposer un objet sur un sommet induit des relations de préséance (mais contrairement au PCHACP, ces relations ne sont pas connues a priori). Par conséquent, les travaux sur le PCHACP, ainsi que ceux portant sur le *Problème du Voyageur de Commerce Asymétrique avec Contraintes de Préséance* ([11], [9]) qui est un problème similaire au PCHACP, sont utiles pour l'élaboration d'un algorithme exact pour la version préemptive du PR (voir chapitre 4).

Chapter 3

A Branch-and-Cut Algorithm for the Non-Preemptive Swapping Problem

Résumé. Ce chapitre analyse la version non préemptive du *Problème de Repositionnement.* Dans cette version les objets sont tous *non déposables* et doivent par conséquent être transportés directement de leur origine vers l'une de leurs destinations potentielles. Après avoir défini certaines propriétés de structure des solutions optimales, un modèle mathématique basé sur des variables binaires est proposé. Un algorithme de coupes et branchements est ensuite présenté pour résoudre de façon exacte la formulation de ce modèle. Les résultats numériques montrent que la formulation est remarquablement serrée dans le sens où la valeur de la relaxation au noeud racine de l'arbre de recherche est très proche de la valeur optimale. Cela permet de résoudre des instances d'assez grande taille en un temps de calcul plutôt court. Les tests numériques mettent également en évidence le fait qu'en général les instances contenant peu de types d'objet (typiquement trois) sont les plus difficiles à résoudre. L'algorithme a été testé avec succès sur des instances géométriques aléatoires contenant jusqu'à 200 sommets et huit types d'objet.

A Branch-and-Cut Algorithm for the Non-Preemptive Swapping Problem

Charles BORDENAVE
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel GENDREAU
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Gilbert LAPORTE
CIRRELT and Canada Research Chair in Distribution Management,
HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine,
Montréal H3T 2A7, Canada.

Article soumis pour publication dans *Naval Research Logistics*

March 2008

Abstract. In the *Swapping Problem* (SP), we are given a complete graph, a set of object types and a vehicle of unit capacity. An initial state specifies the object type currently located at each vertex (at most one type per vertex). A final state describes where these object types must be repositioned. In general there exist several identical objects for a given object type, yielding multiple possible destinations for each object. The SP consists of finding a shortest vehicle route starting and ending at an arbitrary vertex, in such a way that each object is repositioned in its final state. This article exhibits some structural properties of optimal solutions and proposes a branch-and-cut algorithm based on a 0-1 formulation of the problem. Computational results on random instances containing up to 200 vertices and eight object types are reported.

Keywords. Swapping Problem, branch-and-cut, vehicle routing

3.1 Introduction

The *Swapping Problem* (SP) is defined on a complete directed graph $G = (V, A)$, where $V = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) \mid i \in V, j \in V, i \neq j\}$ is the arc set. Without loss of generality, vertex 1 is arbitrarily designated as a *depot*. A cost matrix (c_{ij}) satisfying the triangular inequality is defined on A. We consider a set of m object types $O = \{1, \ldots, m\}$, also referred to as products, located at the vertices. With vertex i is associated a pair (a_i, b_i) of object types corresponding to its supply and its demand. Initially, the supply object is located at the vertex. Each object has a unit weight and appears the same number of times as a supply object and as a demand object. In the SP, the aim is to carry the objects using a unit capacity vehicle, in such a way that all vertices receive their demand object and the total cost is minimized. The vehicle can perform empty trips (called *deadheading*), in which case it is assumed to transport a *null object* denoted by 0. The version of the SP considered in this paper is called *non-preemptive* because objects are *non-droppable*, i.e., they cannot be dropped at temporary locations along the vehicle route. Applications of the SP arise in robotics ([10]) and in printed circuit board assembly ([12]).

The SP was introduced by Anily and Hassin [4] who proved it to be NP-hard by reduction to the *Traveling Salesman Problem* (TSP) and derived interesting structural properties of optimal solutions. They also developed a 2.5-approximation algorithm based on matching and patching methods. The case of a linear graph was analyzed by Anily et al. [2] and was shown to be solvable in $O(n^2)$ time.

A problem closely related to the non-preemptive SP is the *Stacker Crane Problem* (SCP) in which each vertex has a supply or a demand, but not both, and each object appears only once as a supply and as a demand. The SCP is a special case of the asymmetric TSP and also corresponds to a special case of the non-preemptive SP in which there exists only one object for each type. Frederickson et al. [30] analyzed the SCP on a complete graph and proposed a 1.8-approximation heuristic, based on matching and on minimum spanning trees. Atallah and Kosaraju [10] proved that the non-preemptive SCP can be solved in $O(m+nf(n))$ time on a line and in $O(m+n\log n)$

time on a circle, where $f(n)$ is the inverse of Ackermann's function.

Another related problem is the *Bipartite TSP* (BTSP), a TSP in which the number of vertices is even, half white, and half black. The problem consists of determining a shortest Hamiltonian tour in which no two vertices of the same color are consecutive. Chalasani and Motwani [17] have proposed a 2-approximation algorithm for the undirected version of this problem. Ghiani et al. [31] have considered a generalization of the BTSP in which the numbers of white and black vertices may differ, and the number of white vertices between two consecutive black vertices does not exceed a prespecified value. The authors have developed a branch-and-cut algorithm for this NP-hard problem. The BTSP corresponds to a special case of the non-preemptive SP in which there exist only two object types (referred to as the *simple* SP in [4]).

Finally, we mention the *One-Commodity Pickup-and-Delivery TSP* investigated by Hernández-Pérez and Salazar González [35, 36]. In this problem each vertex has a non-negative supply or demand of a single product. The problem is to determine a shortest Hamiltonian tour for a capacitated vehicle in such a way that all requests are satisfied. This problem is NP-hard and was solved by branch-and-cut for up to 100 vertices.

Our aim is to develop, for the first time, an exact branch-and-cut algorithm for the non-preemptive SP on a general graph. The remainder of the paper is organized as follows. In Section 3.2 we prove some structural properties of optimal solutions. An integer linear programming formulation is presented in Section 3.3. Section 3.4 contains a description of the branch-and-cut algorithm. Computational results are presented in Section 3.5, followed by conclusions in Section 3.6.

3.2 Structural properties of optimal solutions

In addition to the notation already introduced, we define the binary coefficients α_{ik} and β_{ik} equal to 1 if and only if $a_i = k$ and $b_i = k$, respectively. Multiple optimal solutions may exist for a given instance, especially when the triangular inequality holds with equality. As in [4] we are interested in solutions having minimum number of arcs

because they induce some interesting structural properties.

Definition 3.2.1. *A solution is called optimal if it has minimum cardinality among all solutions that minimize the objective function.*

Lemma 3.2.1. *In any optimal solution every visited vertex i is incident to exactly one incoming arc carrying object type b_i.*

Proof. Consider an optimal solution S. Suppose there exists a vertex i for which $r \geq 2$ incoming arcs (u_t, i), $t \in \{1, ..., r\}$ carry object type b_i. Then there must exist $r-1$ arcs (i, v_t), $t \in \{1, ..., r-1\}$ exiting i with object type b_i (one object is left at i to satisfy its demand). This means that a non-negative number of b_i objects are temporarily unloaded at i and carried later to another vertex. Since the problem is non-preemptive such drops are not allowed, thus contradicting the optimality of S. □

Lemma 3.2.2. *In any optimal solution every visited vertex i is incident to exactly one outgoing arc carrying object type a_i.*

Proof. Similar to the proof of Lemma 3.2.1 by symmetry. □

Lemma 3.2.3. *In any optimal solution every vertex i satisfying one of the conditions 1) $a_i \neq 0$ and $b_i = 0$, 2) $a_i = 0$ and $b_i \neq 0$, 3) $a_i = b_i = 0$ and $i = 1$, has degree two.*

Proof. Consider a vertex i such that $a_i \neq 0$ and $b_i = 0$. The vehicle must visit i to satisfy its demand. By Lemma 3.2.1 there exists exactly one incoming arc carrying object type $b_i = 0$ (i.e., a deadheading). By Lemma 3.2.2, there exists exactly one outgoing arc carrying object type a_i. Since every object type is non-droppable, no other object can enter i and no other object can exit i. Hence i is a degree-two vertex. The second condition where $a_i = 0$ and $b_i \neq 0$ follows by symmetry. For the third condition, we can also apply Lemmas 3.2.1 and 3.2.2 on i (i.e., the depot) because it is necessarily a visited vertex. Then there exist exactly one incoming arc carrying object type $b_i = 0$ and exactly one outgoing arc carrying $a_i = 0$. Since objects are non-droppable, no other types of arc can be adjacent to i. □

Lemma 3.2.4. *In any optimal solution every vertex is incident to at most one incoming and one outgoing deadheading.*

Proof. Consider an optimal solution S and consider a vertex i visited by S. Two cases are possible: 1) Case 1: $a_i \neq 0$ and $b_i = 0$, $a_i = 0$ and $b_i \neq 0$, $a_i = b_i = 0$ and $i = 1$. By Lemma 3.2.3, i has degree two and therefore i is incident to at most one incoming and one outgoing deadheading. 2) Case 2: $a_i \neq 0$ and $b_i \neq 0$. Suppose there exist $r \geq 2$ incoming deadheadings (u_t, i), $t \in \{1, ..., r\}$. Since i does not require nor supply object type 0, there exist $r \geq 2$ outgoing deadheadings (i, v_t), $t \in \{1, ..., r\}$. By Lemmas 3.2.1 and 3.2.2, i is incident to exactly one incoming arc carrying b_i and exactly one outgoing arc carrying a_i, then $r+1$ arcs enter and exit i. Since $r \geq 2$ there exists at least one pair of deadheadings (u_p, i) and (i, v_q), for some p and $q \in \{1, ..., r\}$, that are traversed consecutively in S and then can be replaced with a single arc (u_p, v_q). This yields a new feasible solution S' that is no worse than S and containing fewer arcs, thus contradicting the optimality of S. □

Lemma 3.2.5. *An optimal solution can possibly pass through a vertex i such that $a_i = b_i$ and different from the depot if and only if $a_i \neq 0$ (or equivalently $b_i \neq 0$). Furthermore if the solution visits such a vertex i, then i is incident to two incoming arcs (one carrying object type b_i and one deadheading) and two outgoing arcs (one carrying object type a_i and one deadheading).*

Proof. We start with the first part of the lemma.

⇒ Suppose there exists a vertex $i \in V \setminus \{1\}$ with $a_i = b_i = 0$ visited in an optimal solution S. By Lemmas 3.2.1 and 3.2.2, there exist exactly one incoming arc (u, i) carrying $b_i = 0$ (i.e., a deadheading) and exactly one outgoing arc (i, v) carrying $a_i = 0$ (i.e., a deadheading). Replacing these two arcs with a unique deadheading (u, v) yields a new feasible solution S' no worse than S and containing fewer arcs, thus contradicting the optimality of S. Therefore if the vehicle can possibly visit such i, then a_i must be non-zero.

⇐ Consider the instance shown in Figure 3.1 containing four vertices located at the corners of the unit square and an additional vertex at the center.

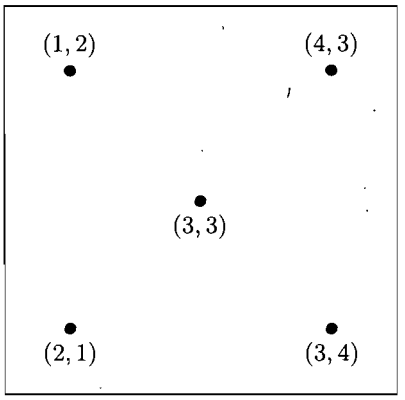


FIG. 3.1 – Instance with a vertex i such that $a_i = b_i$

Object types 1, 2 and 4 have only one source and one destination, and therefore the vehicle must carry them from their origin to their unique destination with no other choice, whereas there exist two alternatives for object type 3 which are shown in Figures 3.2 and 3.3 (one passing through the center point). To construct the corresponding solutions from these two alternatives we need to patch the two connected components together using a pair of deadheadings (Figures 3.4 and 3.5). The total cost of the first solution is 6 whereas the second one has a smaller cost equal to $3 + 2\sqrt{2}$.

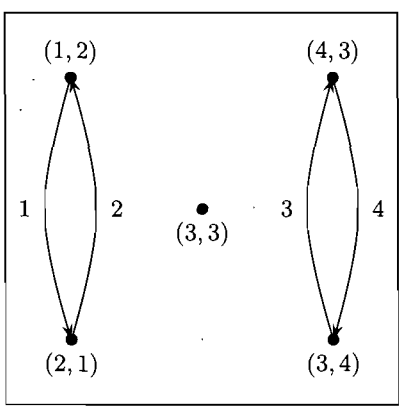


FIG. 3.2 – Alternative 1

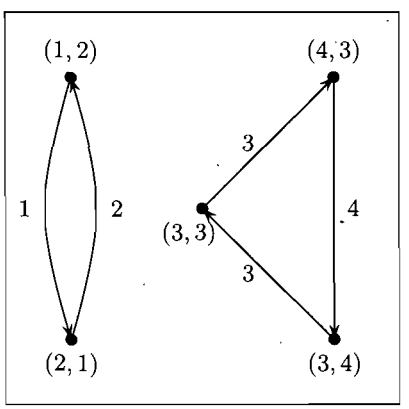


FIG. 3.3 – Alternative 2

To prove the second part of the lemma, consider an optimal solution S visiting a vertex $i \neq 1$ with $a_i = b_i$ and $a_i \neq 0$. Suppose there exists only one arc entering i, a deadheading (u, i). Since $b_i \neq 0$ there exists a deadheading (i, v). Since $a_i = b_i$

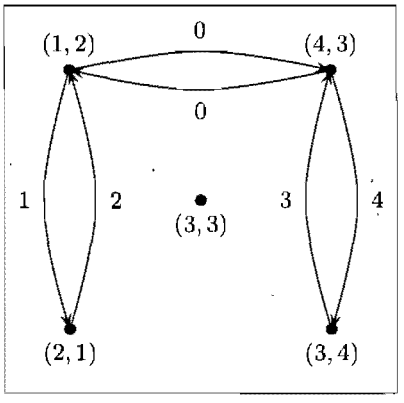


FIG. 3.4 – Solution 1

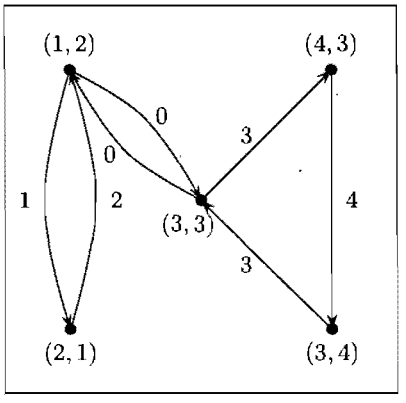


FIG. 3.5 – Solution 2

we can construct a new feasible solution S' by simply replacing these two arcs with a single deadheading (u, v). Again from triangular inequality S' is no worse than S and contains fewer arcs, thus contradicting the optimality of S. Suppose now that this single incoming arc (u, i) carries b_i. The vehicle must exit i with a_i, following an arc (i, v). Since $a_i = b_i$, we can again replace these two arcs with a unique arc (u, v) carrying b_i, thus contradicting the optimality of S. From Lemmas 3.2.1 and 3.2.4, there exist two incoming arcs (an arc carrying b_i and an incoming deadheading) and two outgoing arcs (an arc carrying a_i and an outgoing deadheading). □

Lemma 3.2.5 contradicts an observation by Anily and Hassin [4] which incorrectly states that there always exists an optimal solution for the non-preemptive SP (called pure no-drop case in their paper) that does not visit vertices for which the demand equals the supply.

Theorem 3.2.1. *In any optimal solution each vertex has degree zero, two or four.*

Proof. Lemma 3.2.5 implies that an optimal solution may or may not visit a vertex i with $a_i = b_i$. Therefore such a vertex may have a zero degree in an optimal solution. Every vertex i with $a_i \neq b_i$ must be visited by the vehicle in order to satisfy its demand and supply. By Lemmas 3.2.1 and 3.2.2, every visited vertex is incident to exactly one incoming arc carrying b_i and exactly one outgoing arc carrying a_i, yielding a vertex of degree two. By Lemma 3.2.4, the same vertex can possibly be incident to at most one

additional pair of arcs (incoming and outgoing deadheadings) so that it would become a vertex of degree four. □

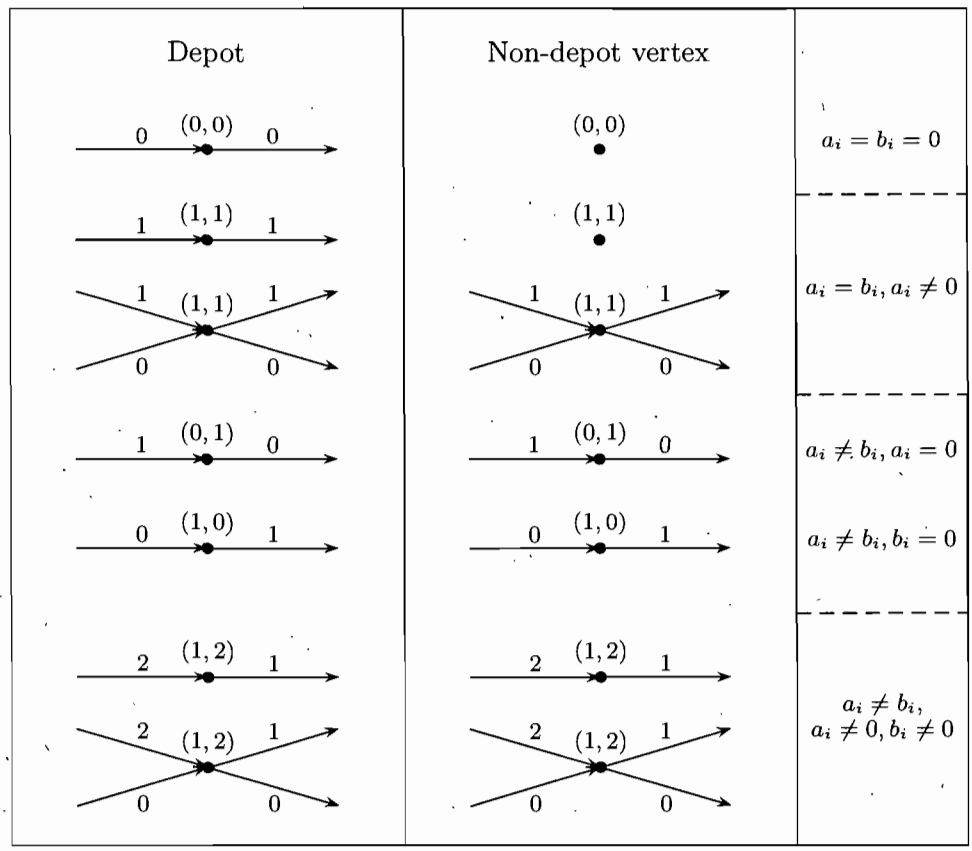


FIG. 3.6 – All possible configurations for a vertex in an optimal solution.

If vertices having the same demand and supply are disallowed (which would force the vehicle to visit each vertex at least once), then the problem can be viewed as a particular case of the *Graphical Traveling Salesman Problem* (GTSP) studied by Cornuéjols et al. [22], a variant of the TSP in which the degree constraints allow each vertex to be visited more than once. If we denote by $GTSP(n)$ and $SP(n)$ the polyhedra associated with the GTSP and this special case of non-preemptive SP respectively, then $SP(n) \subset GTSP(n)$. Therefore every known valid inequality for the GTSP is also valid for this special case of non-preemptive SP.

Definition 3.2.2. *A triplet (i, j, k), $(i, j) \in A$, $k \in O \cup \{0\}$, is called discardable if the transport of object type k from vertex i to vertex j will never be part of any optimal solution. Denote by $\mathcal{N}$ the subset of non-discardable triplets.*

Proposition 3.2.1. *A triplet (i, j, k), $(i, j) \in A$, $k \in O \cup \{0\}$ is discardable if one of the following conditions is satisfied: 1) $a_i = b_i = 0$, $i \neq 1$, 2) $a_j = b_j = 0$, $j \neq 1$, 3) $k \neq a_i$, $k \neq 0$, 4) $k \neq b_j$, $k \neq 0$, 5) $k \neq b_i$, $a_i = 0$, $b_i \neq 0$, 6) $k \neq a_i$, $a_i \neq 0$, $b_i = 0$.*

Proof. The first two conditions are direct consequences of Lemma 3.2.5. The third condition comes from the definition of the problem. Indeed, since object types are non-droppable, the vehicle is not allowed to carry an object of type k from i to j if k is not initially located at i, except possibly for the null object (i.e., (i, j) would be a deadheading). The fourth condition is the symmetric case of the third condition. If $a_i = 0$ and $b_i \neq 0$, then by Lemma 3.2.3, i has degree two in any optimal solution and then, by Lemma 3.2.1, the only object entering i is b_i. By symmetry, from Lemmas 3.2.2 and 3.2.3, if $a_i \neq 0$ and $b_i = 0$ then there exists only one arc exiting i, and this arc carries a_i. This yields the last two conditions. □

To conclude this section, we summarize in Figure 3.6 all possible configurations for a vertex in an optimal solution (in terms of incoming and outgoing arcs). Numerical values have been added for the sake of clarity.

3.3 Mathematical model

The non-preemptive SP can be formulated as an integer linear program. Any feasible solution is a subset of arcs where each one is associated with the object type carried along that arc. For all triplets $(i, j, k) \in \mathcal{N}$, let x_{ij}^k be a binary variable equal to 1 if and only if an object of type k is carried from i to j. The formulation is as follows.

$$\text{minimize} \sum_{(i,j,k)\in N} c_{ij}x_{ij}^{k} \tag{3.1}$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij}^{a_i} = 1 \quad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } a_i = 0 \tag{3.2}$$

$$\sum_{j=1}^{n} x_{ji}^{b_i} = 1 \quad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } b_i = 0 \tag{3.3}$$

$$\sum_{j=1\,|\,b_j=a_i}^{n} x_{ij}^{a_i} = 1 \quad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } a_i \neq 0 \tag{3.4}$$

$$\sum_{j=1\,|\,a_j=b_i}^{n} x_{ji}^{b_i} = 1 \quad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } b_i \neq 0 \tag{3.5}$$

$$\sum_{j=1\,|\,b_j=a_i}^{n} x_{ij}^{a_i} + \sum_{j=1}^{n} x_{ij}^{0} \geq 1 \quad , a_i = b_i \text{ and } i = 1 \tag{3.6}$$

$$\sum_{j=1\,|\,a_j=b_i}^{n} x_{ji}^{b_i} + \sum_{j=1}^{n} x_{ji}^{0} \geq 1 \quad , a_i = b_i \text{ and } i = 1 \tag{3.7}$$

$$\sum_{j=1}^{n}(x_{ij}^{k} - x_{ji}^{k}) = \alpha_{ik} - \beta_{ik} \quad , \forall\, i \in V, \forall k \in O \cup \{0\} \tag{3.8}$$

$$\sum_{j=1}^{n}(x_{ji}^{0} - x_{ji}^{b_i}) = 0 \quad , \forall\, i \in V \text{ s.t. } a_i = b_i \text{ and } i \neq 1 \tag{3.9}$$

$$\sum_{k=0}^{m}\sum_{i\in U}\sum_{j\notin U} x_{ij}^{k} \geq 1 \quad , \forall\, U \in \mathcal{U} \tag{3.10}$$

$$x_{ij}^{k} \in \{0,1\} \quad , \forall\,(i,j,k) \in \mathcal{N}, \tag{3.11}$$

where $\mathcal{U}$ is a family of vertex sets to be defined in Section 3.3.3, and $\mathcal{N}$ is the set of possibly optimal arcs (see Definition 3.2.2). We now explain the various constraints used in the formulation.

3.3.1 Supply and demand constraints

If i is a vertex for which $a_i \neq b_i$, then an arc carrying a_i must leave i. There are two cases: either the supply of the given vertex is null and therefore the vehicle can carry this object to any other vertex, or the vertex has a non-zero supply and then

the supplied object type must be carried to a vertex requiring that type. In both cases, by Lemma 3.2.2, there exists exactly one outgoing arc carrying object type a_i (constraints 3.2 and 3.4). If a given vertex has the same supply and demand, then it is not certain that the vertex will be part of an optimal solution, except if the vertex is the depot (see Lemma 3.2.5). In this case, the vehicle can possibly load the supplied object and carry it to a vertex requiring that object type, and it may also leave the vertex with a deadheading (see Lemmas 3.2.2 and 3.2.4) (constraints 3.6). The constraints 3.3, 3.5 and 3.7 associated with the demand follow by symmetry.

3.3.2 Flow conservation constraints

Constraints 3.8 represent the flow conservation constraints. They are similar to the constraints proposed in [4] and take care of the conservation of objects at the vertices. Relaxing the integrality requirement may yield solutions satisfying constraints 3.8 but violating Lemma 3.2.5. Such a fractional solution is depicted in Figure 3.7. The numbers on the arcs show the object type carried and the associated x_{ij}^k values in boldface. Therefore we consider constraints 3.9 which ensure that if a deadheading enters a vertex $i \neq 1$ with $a_i = b_i$, then an arc carrying the demand b_i also enters i with the same flow.

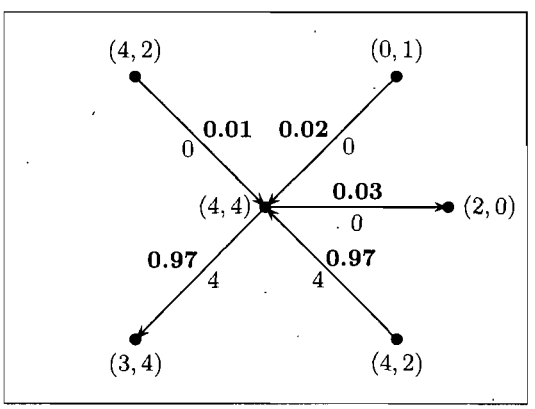


FIG. 3.7 – Part of solution satisfying constraint 3.8 but violating constraint 3.9

3.3.3 Subtour elimination constraints

Standard subtour elimination constraints (SEC) for the directed TSP state that in any optimal solution the vehicle must leave (and equivalently enter) every proper subset of vertices at least once. As we have seen, in the non-preeemptive SP the vertices $i \in V \setminus \{1\}$ with $a_i = b_i$ may or may not be part of an optimal solution. Therefore standard SEC must be slightly modified. Actually, subtour elimination constraints 3.10 are the same as in the directed TSP but the subset of vertices to which the constraint can be applied must satisfy additional requirements.

Definition 3.3.1. *A subset $U \subset V$ is called SEC-compatible if it is valid to impose an SEC on U. Denote by $\mathcal{U}$ the family of SEC-compatible subsets.*

Proposition 3.3.1. *$U \in \mathcal{U}$ if and only if it satisfies one of the two conditions 1) $1 \in U$ and $\exists\, i \in V \setminus U$ such that $a_i \neq b_i$, or 2) $1 \notin U$ and $\exists\, i \in U$ such that $a_i \neq b_i$.*

Proof. $\Rightarrow$ Let $U \in \mathcal{U}$. The vehicle must cross the border of U (in both directions) at least once. Therefore there must exist a vertex in U and a vertex in $V \setminus U$ visited in any optimal solution. Lemma 3.2.5 shows that not all vertices are necessarily part of the solution but by definition any solution passes through vertices i with $a_i \neq b_i$ (to satisfy their supply and demand) and also, by definition, any solution must visit the depot. These are the two types of vertices that are guarantied to be visited. If the depot is in U, then there exists a vertex i such that $a_i \neq b_i$ that is not in U. Otherwise, if the depot is not in U, then there exists a vertex i in U such $a_i \neq b_i$.
$\Leftarrow$ Let $U \subset V$. Suppose that the depot belongs to U and that there exists a vertex $i \in V \setminus U$ such that $a_i \neq b_i$. By definition, every feasible solution must visit the depot and also the vertex i to satisfy its supply and demand. Therefore the solution must contain an arc from U to $V \setminus U$, which is an SEC on U. Replacing U by its complementary $V \setminus U$ gives the second condition. $\square$

3.3.4 Comb inequalities

Comb inequalities are not part of the basic formulation but they will be dynamically incorporated into the model during the branch-and-cut algorithm. These inequalities were first identified by Chvátal [19] and then generalized by Grötschel and Padberg [33]. A comb is defined by a *handle* $H \subset V$, and an odd number $t \geq 3$ of *teeth* $T_i \subset V$ $(i = 1, \ldots, t)$ such that (see Figure 3.8):

$$\begin{aligned} H \cap T_i &\neq \emptyset\,, \quad (i = 1, \ldots, t) \\ T_i \setminus H &\neq \emptyset\,, \quad (i = 1, \ldots, t) \\ T_i \cap T_j &\neq \emptyset\,, \quad (1 \leq i < j \leq t). \end{aligned}$$

For the TSP the comb inequality is the following constraint:

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq 3t + 1,$$

where the notation $\delta(S)$ is the set of edges having only one endpoint in S, and $x(S)$ represents the sum of the values of the edges having their two endpoints in S.

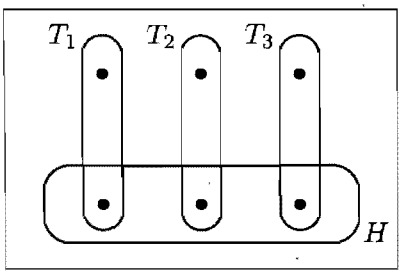


FIG. 3.8 – Minimal comb configuration

In the non-preemptive SP the graph is directed and in a solution each arc is associated with an object type. Simply replacing each arc (i, j) by an edge $e = (i, j)$ with value $x_e = \sum_{k=0}^{m}(x_{ij}^k + x_{ji}^k)$ is not sufficient to define a valid comb inequality for the non-preemptive SP. We have to take into account the fact that some vertices are not necessarily required to be part of the solution (i.e., vertices i with $a_i = b_i$). The subsets H and T_i must satisfy an additional requirement which is introduced in the next definition.

Definition 3.3.2. *A subset $H \subset V$ and an odd number $t \geq 3$ of subsets $T_i \subset V$ $(i = 1, \ldots, t)$ such that 1) $H \cap T_i \neq \emptyset$, $(i = 1, \ldots, t)$, 2) $T_i \setminus H \neq \emptyset$, $(i = 1, \ldots, t)$ and 3) $T_i \cap T_j \neq \emptyset$, $(1 \leq i < j \leq t)$, is said to be comb-compatible if it is valid to impose a comb inequality from the handle H and the teeth T_i.*

Proposition 3.3.2. *A comb (H, T_i) is comb-compatible if the following two conditions are satisfied for each T_i: 1) $1 \in T_i \setminus H$ or $\exists\, i \in T_i \setminus H$ such that $a_i \neq b_i$, 2) $1 \in T_i \cap H$ or $\exists\, i \in T_i \cap H$ such that $a_i \neq b_i$.*

Proof. The proof is similar to that of Proposition 3.3.1. If a vertex has the same demand and supply and is not the depot, the vehicle can possibly skip it in an optimal solution. Such a vertex is somehow unconstrained as its supply can satisfy its own demand. On the other hand, the depot and vertices i with $a_i \neq b_i$ must be visited at least once in any solution. Since the minimal comb configuration (Figure 3.8) consists of exactly one vertex in each $T_i \setminus H$ and $T_i \cap H$, having the depot or a vertex i with $a_i \neq b_i$ in each of these subsets forces the vehicle to cross each border an appropriate number of times to satisfy the standard comb inequality. □

3.4 Branch-and-cut algorithm

We have developed a branch-and-cut algorithm for the non-preemptive SP. Initially, the subtour elimination constraints and the integrality constraints are relaxed. We denote the current linear program by LP.

Whenever the relaxation is solved, an attempt is made to detect some violated inequalities of a certain type. If some are detected, they are added to the current relaxation which is solved again. The process continues until no more violated inequalities can be identified. At this point, if the optimal LP solution satisfies the integrality constraints, it becomes the new incumbent. Otherwise, branching is performed by creating two child nodes and adding them to the node set of the branch-and-cut tree which is referred to as *NODEPOOL* (see the following pseudo-code of *NodeTreatment*).

Procedure 3.4.1 *NodeTreatment*
Input: *LP*, *NODEPOOL*

1: $detectedCut$ = **true**
2: **while** $detectedCut$ **do**
3: **while** $FINDSUBTOUR$ **do**
4: $detectedCut$ = **true**
5: $ADDSUBTOUR$
6: $SOLVE$
7: **end while**
8: **if** $FINDCOMBS$ **then**
9: $detectedCut$ = **true**
10: $ADDCOMBS$
11: $SOLVE$
12: **else**
13: $detectedCut$ = **false**
14: **end if**
15: **end while**
16: **if** $ISINT$ **then**
17: $incumbent \leftarrow x^*$
18: **else**
19: create $leftChild$
20: create $rightChild$
21: $NODEPOOL \leftarrow NODEPOOL \cup \{leftChild\}$
22: $NODEPOOL \leftarrow NODEPOOL \cup \{rightChild\}$
23: **end if**

To separate subtour elimination constraints we use the Hao-Orlin algorithm ([34]) to compute a global minimum cut in the support graph of the current solution. If the value of the minimum cut is strictly less than 1, we check for SEC-compatibility as defined in Section 3.3.3. If the cutset U is SEC-compatible, then the corresponding SEC constraint is added to the LP which is solved again.

We have used two different heuristics to separate comb inequalities. The first one comes from the publicly available package CVRPSEP developed by Lysgaard ([41]) for the *Capacitated Vehicle Routing Problem* ([40, 42]), and the second one is our implementation of the heuristic proposed by Naddef and Thienel [45]. The two heuristics are executed subsequently and each new cut is stored (in our implementation we limit the number of stored cuts to 30 for a given input LP). Cuts for which comb-compatibility is satisfied (see Proposition 3.3.2) are added to the LP which is solved again.

We have applied the standard branching on variables. Given the current fractional solution x^*, we select the variable closest to 0.5. The lexicographic order is used to break ties. Then we generate two child nodes (*leftChild* and *rightChild*, see the pseudo-code of *NodeTreatment*) by fixing the value of that variable to either 0 or 1. These two new nodes are added to the pool (*NODEPOOL*) which maintains the nodes of the branch-and-cut tree as a priority stack. In our computation the best-bound search strategy was used to explore the pool. We have tested several branching rules (branching on variable with various selection rules, branching on cutset) but the standard branching rule has, in general, produced the best results.

3.5 Computational results

The branch-and-cut algorithm was coded in C++ and integrated in a branch-and-bound framework called *OOBB*, which stands for Object-Oriented Tools for Parallel Branch-and-Bound, currently in development at the CIRRELT in Montreal. Our code uses the sequential mode. As for the LP solver we used ILOG CPLEX 10.1. Tests were performed on an AMD Opteron Dual Core 285 2.6GHz running Linux.

To generate the instances, vertices were randomly distributed in the 500×500 square according to a discrete distribution. We have associated to the vertices a random supply and a random demand within $\{0, \ldots, m\}$ such that each object type was requested and supplied at least once. Vertices $i \neq 1$ with $a_i = b_i = 0$ were not generated since it has been proved in Section 2 that they are not visited in an optimal solution. We

have considered instances with $50 \leq n \leq 200$ vertices (with an increment of 10) and $3 \leq m \leq 8$ object types. For each pair (n, m) three different instances were generated, yielding a total of 96 instances. Computational results given in this paper represent the average results over these three sets of instances. The running time limit was set to 5400 seconds.

Our results are summarized in Table 3.1 where we report the average number of cuts added to the relaxation, the average number of nodes in the branch-and-cut tree and the average running time in seconds. All instances have been solved to optimality, and 31% were solved at the root node. In general, instances with few object types (typically 3 or 4) were the hardest to solve. This can be explained by the fact that the number of potential destinations (for each object) increases as the number of object types decreases.

	Cuts		B&C Nodes		Seconds	
$n \setminus m$	3,4,5	6,7,8	3,4,5	6,7,8	3,4,5	6,7,8
50, 60, 70, 80	59	8	6	1	7	1
90, 100, 110, 120	158	95	8	6	79	79
130, 140, 150, 160	212	112	11	17	239	204
170, 180, 190, 200	378	205	12	7	955	580

TAB. 3.1 – Summary of computational results on random instances

Table 3.2 gives the average relative gap at the root which is the relative difference between the value of the optimal solution and the value obtained at the first node of the branch-and-cut tree. As we can see the gap was extremely small.

Tables 3.3, 3.4, 3.5 and 3.6 show the results of our branch-and-cut algorithm on all instances (for each pair (n, m) the reported results correspond to the average over three different instances). The column headings are as follows. **Subtours** is the number of subtour elimination constraints added to the relaxation, **Combs** is the number of comb inequalities added to the relaxation, **Gap** is the relative gap between the solution value obtained at the root node of the search tree and the optimal solution value, **Nodes**

$n \setminus m$	3	4	5	6	7	8
50, 60, 70, 80	0.0033	0.0011	0.0008	0.0000	0.0000	0.0000
90, 100, 110, 120	0.0006	0.0008	0.0001	0.0002	0.0001	0.0004
130, 140, 150, 160	0.0011	0.0003	0.0002	0.0006	0.0004	0.0002
170, 180, 190, 200	0.0003	0.0002	0.0002	0.0000	0.0004	0.0004

TAB. 3.2 – Relative gap at the root of the branch-and-cut tree

is the total number of nodes in the branch-and-cut tree, and **Seconds** is the running time in seconds.

3.6 Conclusions

We have proposed the first ever exact algorithm for the non-preemptive Swapping Problem on a complete graph. We have elaborated a mathematical model based on typical arc-routing variables as well as a branch-and-cut algorithm to solve it. Computational tests show our algorithm can solve reasonably large instances to optimality within acceptable computation times.

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(50,3)	60	23	0.0125	7	4
(50,4)	24	0	0.0000	1	0
(50,5)	28	4	0.0032	7	2
(50,6)	5	0	0.0000	3	0
(50,7)	3	0	0.0000	1	0
(50,8)	7	0	0.0000	1	0
(60,3)	40	0	0.0000	1	2
(60,4)	31	4	0.0038	3	2
(60,5)	21	0	0.0001	5	1
(60,6)	1	0	0.0000	1	0
(60,7)	5	0	0.0000	1	0
(60,8)	1	0	0.0000	1	0
(70,3)	32	73	0.0008	9	12
(70,4)	43	33	0.0005	37	19
(70,5)	39	0	0.0000	1	4
(70,6)	15	0	0.0000	1	1
(70,7)	11	0	0.0000	1	1
(70,8)	1	0	0.0000	1	0
(80,3)	62	46	0.0000	3	14
(80,4)	70	58	0.0002	3	24
(80,5)	14	2	0.0000	1	2
(80,6)	31	0	0.0000	1	5
(80,7)	17	0	0.0000	1	3
(80,8)	1	0	0.0000	1	0

TAB. 3.3 – Detailed computational results on random instances ($50 \leq n \leq 80$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(90,3)	215	22	0.0001	7	75
(90,4)	195	24	0.0001	3	87
(90,5)	153	42	0.0049	19	84
(90,6)	7	0	0.0000	5	2
(90,7)	22	0	0.0003	9	9
(90,8)	139	20	0.0015	3	78
(100,3)	164	36	0.0012	7	70
(100,4)	34	52	0.0018	13	33
(100,5)	84	0	0.0001	5	34
(100,6)	20	0	0.0000	1	6
(100,7)	33	4	0.0001	11	21
(100,8)	160	0	0.0000	1	132
(110,3)	42	36	0.0000	7	21
(110,4)	252	71	0.0013	11	233
(110,5)	13	0	0.0000	1	4
(110,6)	182	91	0.0003	9	242
(110,7)	20	0	0.0000	1	7
(110,8)	31	0	0.0003	5	16
(120,3)	194	168	0.0010	23	269
(120,4)	66	0	0.0000	1	25
(120,5)	29	4	0.0000	3	12
(120,6)	251	12	0.0007	29	339
(120,7)	51	0	0.0000	1	28
(120,8)	100	0	0.0000	3	73

TAB. 3.4 – Detailed computational results on random instances ($90 \leq n \leq 120$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(130,3)	119	151	0.0013	33	274
(130,4)	51	0	0.0000	3	20
(130,5)	89	80	0.0003	19	129
(130,6)	62	90	0.0024	25	187
(130,7)	35	4	0.0003	3	26
(130,8)	69	55	0.0001	15	148
(140,3)	158	33	0.0018	3	121
(140,4)	154	184	0.0002	7	266
(140,5)	46	0	0.0000	1	31
(140,6)	31	5	0.0000	3	23
(140,7)	48	2	0.0000	3	43
(140,8)	41	33	0.0001	7	81
(150,3)	459	47	0.0003	11	882
(150,4)	162	42	0.0010	9	211
(150,5)	189	38	0.0006	27	346
(150,6)	79	0	0.0000	1	67
(150,7)	206	55	0.0013	29	576
(150,8)	59	0	0.0000	1	59
(160,3)	179	164	0.0012	19	418
(160,4)	119	51	0.0000	3	143
(160,5)	34	0	0.0000	1	23
(160,6)	155	42	0.0000	3	260
(160,7)	57	0	0.0002	3	64
(160,8)	78	133	0.0006	111	909

TAB. 3.5 – Detailed computational results on random instances ($130 \leq n \leq 160$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(170,3)	569	101	0.0001	5	1595
(170,4)	200	44	0.0006	15	354
(170,5)	315	50	0.0002	9	816
(170,6)	56	0	0.0000	1	57
(170,7)	235	189	0.0006	35	1467
(170,8)	116	137	0.0009	13	631
(180,3)	420	162	0.0005	11	1106
(180,4)	356	1	0.0002	3	646
(180,5)	53	71	0.0003	9	145
(180,6)	166	122	0.0001	7	487
(180,7)	291	54	0.0010	9	1097
(180,8)	45	0	0.0000	1	56
(190,3)	111	50	0.0002	5	180
(190,4)	133	69	0.0001	21	312
(190,5)	237	7	0.0001	5	589
(190,6)	99	0	0.0000	1	135
(190,7)	49	1	0.0000	1	69
(190,8)	31	7	0.0001	9	74
(200,3)	538	257	0.0006	47	3251
(200,4)	424	14	0.0000	3	1555
(200,5)	292	59	0.0001	7	911
(200,6)	247	0	0.0000	1	800
(200,7)	296	5	0.0000	3	886
(200,8)	309	10	0.0005	3	1207

TAB. 3.6 – Detailed computational results on random instances ($170 \leq n \leq 200$)

Chapter 4

A Branch-and-Cut Algorithm for the Preemptive Swapping Problem

Résumé. Ce chapitre porte sur la version préemptive du *Problème de Repositionnement.* Dans cette version les objets sont tous *déposables* et sont par conséquent autorisés à être temporairement déchargés sur des sommets intermédiaires avant d'être transportés vers l'une de leurs destinations potentielles. Cette version est nettement plus difficile à résoudre que la version non préemptive étudiée au chapitre 3 puisqu'il faut prendre en considération les relations de préséance induites par le fait de déposer temporairement un objet sur un sommet. Après avoir défini certaines propriétés des solutions optimales et présenté une technique pour gérer le problème de la préemption, on présente un modèle mathématique basé sur des variables binaires et des variables continues. On décrit ensuite un algorithme de coupes et branchements que l'on a développé pour résoudre de façon exacte la formulation de ce modèle. Les résultats numériques montrent que le modèle est serré puisque la valeur de solution au noeud racine de l'arbre de recherche est en général très proche de la valeur optimale. Cet algorithme a permis de résoudre optimalement des instances géométriques aléatoires contenant jusqu'à 100 sommets et huit types d'objet. Contrairement à la version non préemptive, ici le nombre de types d'objet ne semble pas influencer les temps de calcul.

A Branch-and-Cut Algorithm for the Preemptive Swapping Problem

Charles BORDENAVE
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel GENDREAU
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Gilbert LAPORTE
CIRRELT and Canada Research Chair in Distribution Management,
HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine,
Montréal H3T 2A7, Canada.

Article soumis pour publication dans *Mathematical Programming*

April 2008

Abstract. In the *Swapping Problem* (SP), every vertex of a complete graph may supply and demand an object of a known type. A vehicle of unit capacity starting and ending its tour at an arbitrary vertex is available for carrying objects of given types between vertices. The SP consists of determining a minimum cost route that allows the vehicle to satisfy every supply and demand. This article investigates the *preemptive* version of the SP in which the objects are allowed to be dropped at temporary locations along the route. The problem is modeled as a mixed integer linear program which is solved by branch-and-cut. Computational results on random geometric instances containing up to 100 vertices and eight object types are reported.

Keywords. Swapping Problem, vehicle routing, branch-and-cut, precedence relationships

4.1 Introduction

The purpose of this paper is to describe a branch-and-cut algorithm for the *Swapping Problem* (SP) defined as follows. Let $G = (V, A)$ be a complete directed graph, where $V = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) \mid i \in V, j \in V, i \neq j\}$ is the arc set. Without loss of generality, vertex 1 is arbitrarily designated as a *depot*. A cost matrix (c_{ij}) satisfying the triangular inequality is defined on A. We consider a set $O = \{1, \ldots, m\}$ of m *object types* located at the vertices. With vertex i is associated a pair (a_i, b_i) of object types corresponding to its supply and its demand. Initially, the supply object is located at the vertex. Each object has a unit weight and appears the same number of times as a supply object and as a demand object. In the SP, the aim is to carry the objects using a unit capacity vehicle, in such a way that all vertices receive their demand object and the total cost is minimized. The vehicle can perform empty trips (called *deadheading*), in which case it is assumed to carry a *null object* denoted by 0. The version of the SP considered in this paper is called *preemptive*, meaning that the objects are *droppable*, i.e., they can be dropped at temporary locations along the route before being moved to their final destination.

The SP was introduced by Anily and Hassin [4] who proved it is NP-hard by reduction to the *Traveling Salesman Problem* (TSP), and derived interesting structural properties of optimal solutions. They also developed a 2.5-approximation algorithm based on matching and patching methods. The case of a linear graph was analyzed by Anily et al. [2] and was shown to be solvable in $O(n^2)$ time. The same authors ([3]) proved that the preemptive SP on a tree is NP-hard by reduction to the *Steiner Tree Problem* on a bipartite graph, but the case where $m = 2$ can be solved in polynomial time. The authors developed a 1.5-approximation heuristic for the case where $m \geq 3$.

A well known problem closely related to the SP is the *Stacker Crane Problem* (SCP) in which each vertex has a supply or a demand, but not both, and each object appears only once as a supply and as a demand. This NP-hard problem is a special case of the asymmetric TSP and also corresponds to a special case of the SP in which there exists only one object for each type. Atallah and Kosaraju [10] proved that the preemptive

SCP on a line and on a circle can be solved in $O(m+n)$ time. Frederickson and Guan [28] showed that the preemptive SCP on a tree can also be solved in polynomial time, and proposed two exact algorithms of order $O(m + n \log n)$ and $O(m + qn)$ (where $q \leq \min\{m, n\}$). Table 4.1 summarizes known complexity results for the preemptive versions of the SCP and the SP.

Graph structure	Preemptive SCP	Preemptive SP
General	NP-hard[1]	NP-hard [4]
Tree	Polynomial [28]	NP-hard [3]
Circle	Polynomial [10]	?
Line	Polynomial [10]	Polynomial [2]

[1]By reduction to the TSP

TAB. 4.1 – Complexity results for the preemptive SCP and the preemptive SP

Another related problem is the *One-commodity Pickup-and-Delivery TSP* (m-PDTSP when m commodities are considered) investigated by Hernández-Pérez and Salazar González [35, 36]. In the 1-PDSTP, each vertex of a complete graph is associated with a non-negative demand or supply of a single product. The problem consists of determining a shortest Hamiltonian tour for a capacitated vehicle, starting and ending its route at an arbitrary vertex, in such a way that every request is satisfied. The 1-PDTSP and the SP belong to the class of the *many-to-many* routing problems ([14]), in which multiple supplies can serve multiple demands. The authors have proposed a branch-and-cut algorithm that combines standard TSP constraints with Benders cuts, clique cuts and multistar inequalities (see [40] and [42]). Optimal solutions on instances containing up to 100 vertices were reported.

Finally, routing problems with precedence constraints are also related to the preemptive SP. These include the *Sequential Ordering Problem* (SOP) introduced by Escudero [27]. The SOP is defined on a complete directed graph $G = (V, A)$, where the vertex set V represents jobs to be processed on a single machine, and the arc set A represents sequencing of the jobs. A cost matrix representing processing and setup times is defined on A. Given an additional precedence graph that specifies sequencing

relationships between the different jobs, the SOP consists of determining a minimum cost Hamiltonian path in which no precedence constraints are violated. Ascheuer et al. [7, 8], and Ascheuer [6] have proposed branch-and-cut algorithms that make use of various types of valid inequalities for solving this NP-hard problem. It should be noted that in the preemptive SP, unlike in the SOP, the precedence relationships are not known a priori, but they are induced by the solution.

Our aim is to develop, for the first time, an exact branch-and-cut algorithm for the preemptive SP on a general graph. The remainder of the paper is organized as follows. In Section 4.2 we prove some properties of optimal solutions and present our approach to handle preemption. Sections 4.3 and 4.4 cover the mathematical model and the branch-and-cut algorithm. Computational results are presented in Section 4.5. A description of how the algorithm can be modified to solve the general or *mixed* SP is given in Section 4.6, followed by conclusions in Section 4.7.

4.2 Properties of optimal solutions

After recalling an important result presented in [4] and exhibiting some structural properties of optimal solutions, we introduce our technique for the handling of precedence relationships induced by droppable objects. We use the same definition of optimality as in [4].

Definition 4.2.1. *A solution is called optimal if it has minimum cardinality among all solutions that minimize the objective function.*

4.2.1 Structural properties

Theorem 4.2.1 (Anily and Hassin [4]). *There exists an optimal solution in which every vertex $i \in V$ satisfies the following properties: 1) i is incident to at most one incoming arc carrying an object of type b_i, one outgoing arc carrying an object of type a_i, and at most one additional pair of deadheadings (one entering i, the other leaving*

it); 2) if there is a drop at i, then i is different from the depot and the drop is associated with the first entry in i and the last exit from it.

Theorem 4.2.1 is the backbone of our model. It implies that there exists an optimal solution in which every vertex is visited at most three times. It also indicates the possible order in which the vehicle traverses the arcs incident to a given vertex. In the remainder of this section we will use this theorem to describe all possible configurations that can be part of an optimal solution.

Proposition 4.2.1. *In an optimal solution the depot can be possibly visited twice if and only if $a_1 \neq 0$ and $b_1 \neq 0$.*

Proof. Consider an optimal solution S.
$\Rightarrow$ Suppose the depot is visited twice (i.e., it is incident to four arcs), and $a_1 = 0$ or $b_1 = 0$ (or both). By Theorem 4.2.1, there exists at most one incoming arc carrying b_1 and one outgoing arc carrying a_1 (actually since the depot must necessarily be visited at least once in any solution, "at most once" means "exactly once"). Therefore no additional null object can be carried to or from the depot. Furthermore, by Theorem 4.2.1, there is no drop at the depot. Then the depot is incident to two arcs, thus contradicting our assumption.
$\Leftarrow$ Suppose $a_1 \neq 0$ and $b_1 \neq 0$ like in the instance shown in Figure 4.1 (the depot is represented as a square dot). On such a small instance it is easy to check by enumeration that the optimal solution is the one depicted in Figure 4.2. The depot is incident to four arcs: a pair associated with the carrying of its demand and supply and a pair of deadheadings (incoming and outgoing). □

Definition 4.2.2. *A vertex i with $a_i = b_i$ is called a transshipment vertex.*

Proposition 4.2.2. *In an optimal solution no transshipment vertex different from the depot can be visited exactly once.*

Proof. Consider an optimal solution S. Suppose there exists a transshipment vertex $i \in V \setminus \{1\}$ that is visited exactly once. Then there exists an incoming arc (u, i)

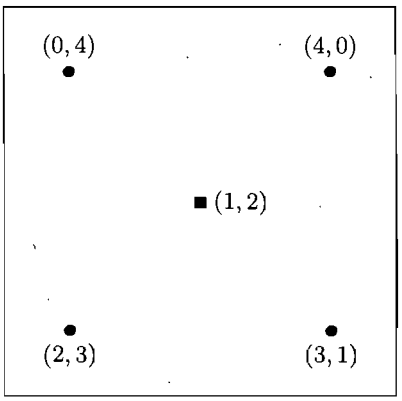


FIG. 4.1 – Instance with $a_1 \neq 0$ and $b_1 \neq 0$

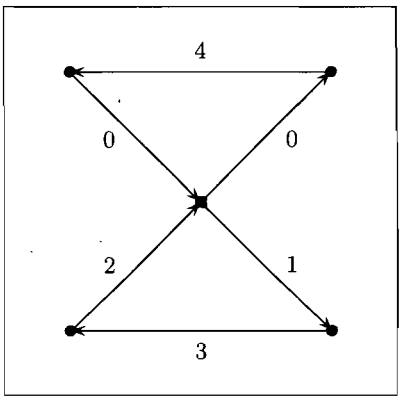


FIG. 4.2 – Optimal solution

carrying object type b_i and an outgoing arc (i, v) carrying a_i. By Definition 4.2.2, $a_i = b_i$. Replacing these two arcs with a unique arc (u, v) carrying $a_i = b_i$, yields a new feasible solution S' which is no worse than S and contains fewer arcs, thus contradicting the optimality of S. The proposition does not hold for the depot because by definition the depot belongs to all feasible solutions and therefore cannot be skipped by the vehicle (the contraction described above cannot be applied). □

Proposition 4.2.3. *If object type k is dropped at vertex i, then $k \neq a_i$ and $k \neq b_i$.*

Proof. By Theorem 4.2.1, there exists at most one incoming arc carrying b_i and one outgoing arc carrying a_i. Therefore the dropped object, if any, must be different from a_i and b_i. □

Proposition 4.2.4. *There exists an optimal solution that does not contain two consecutive arcs associated with the same object type.*

Proof. Follows from the triangular inequality of the cost matrix and Definition 4.2.1. □

The structural properties of optimal solutions are summarized in Figure 4.3 which depicts all possible configurations for a vertex, in terms of incoming and outgoing arcs, in an optimal solution. Numerical values have been added for the sake of clarity. When

an object of type k is used, it is assumed to be different from the null object, from the supply, and from the demand (i.e., $k \neq 0$, $k \neq a_i$ and $k \neq b_i$).

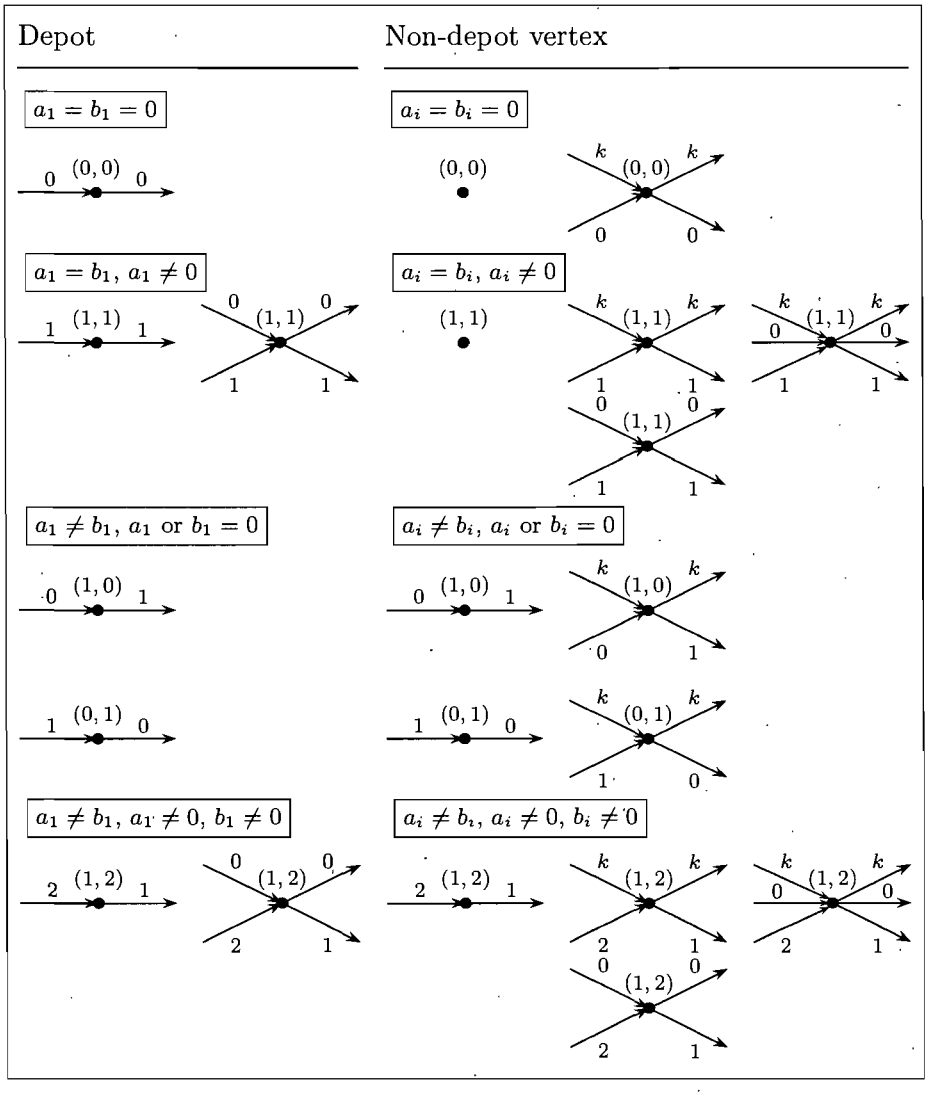


FIG. 4.3 – All possible configurations for a vertex in an optimal solution

Proposition 4.2.5. *The choice of the depot may influence the value of the optimal solution.*

Proof. Consider the instance and solution shown in Figure 4.4 in which the depot is located at the center of the unit square. On such a small instance it is easy to check by

enumeration that the solution is optimal and has a cost of 6.4. Now if we choose the upper-left vertex as the depot, the optimal solution has a cost of 5.4 (see Figure 4.5). □

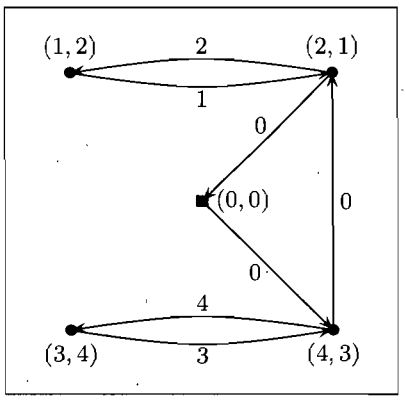


FIG. 4.4 – Optimal solution of cost 6.4

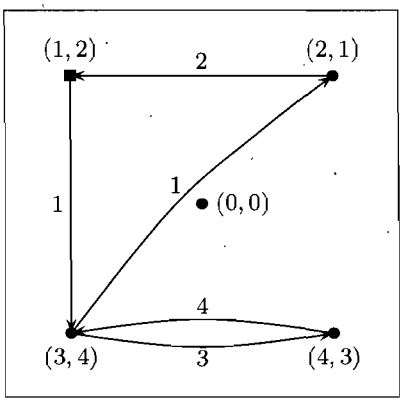


FIG. 4.5 – Optimal solution of cost 5.4

4.2.2 Handling preemption

Handling preemption generates a significant level of difficulty to the formulation of the SP since it is necessary to keep information on the portion of the vehicle route prior to visiting a vertex. This stems from the fact that the vehicle cannot reload an object at a vertex if the object has not been previously dropped at that vertex. We must somehow keep track of the vertices already visited. To this end, we split each vertex into three vertices i, $i+n$ and $i+2n$ such that $a_i = a_{i+n} = a_{i+2n}$, $b_i = b_{i+n} = b_{i+2n}$ and $c_{i,i+n} = c_{i,i+2n} = c_{i+n,i+2n} = 0$ (see Figures 4.6 and 4.7). This triplication is justified by Theorem 4.2.1 which implies that the vehicle may visit the same vertex at most three times in an optimal solution. In this representation, a vertex i is used for a first visit, whereas $i+n$ and $i+2n$ represent a second and third visit to i respectively. This precedence relationship is denoted by the symbol $\prec$. For example $i \prec i+n$ means that vertex i must be visited prior $i+n$ in any feasible solution.

In what follows we use the sets $V_1 = \{1, \ldots, n\}$, $V_2 = \{n+1, \ldots, 2n\}$ and $V_3 = \{2n+1, \ldots, 3n\}$ to distinguish between the triplicates. The full set of vertices is denoted by $\mathcal{V} = V_1 \cup V_2 \cup V_3$. To avoid confusion between the vertices, a vertex of V is called

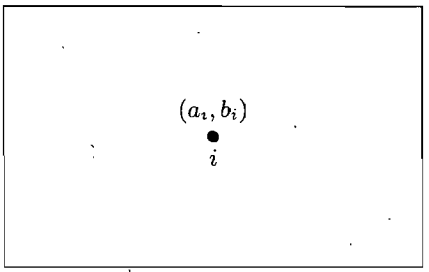


FIG. 4.6 – Original vertex

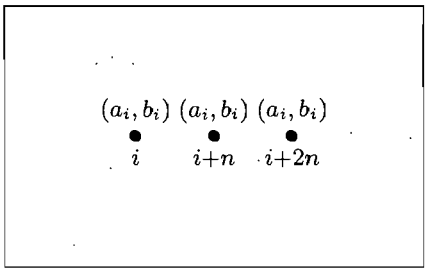


FIG. 4.7 – Vertex triplication (copies)

the *original* vertex and the triplicates are referred to as its *copies*. We also consider an additional object, noted -1, that can only be carried between two consecutive copies of the same vertex (i.e., from i to $i+n$ and from $i+n$ to $i+2n$). More precisely, if the vehicle visits an original vertex three times, then after triplication object -1 is not used among the arcs incident to its copies. Otherwise, -1 is carried from i to $i+n$ and from $i+n$ to $i+2n$ when the original vertex is visited only once, and from $i+n$ to $i+2n$ when the original vertex is visited twice. This representation will allow us to introduce some constraints to handle preemption, i.e., to force the vehicle to always visit the first copy before the second copy, and the second copy before the third one.

4.2.3 Discardable arcs

From Theorem 4.2.1 many arcs associated with an object type can be discarded because there exists an optimal solution that does not contain them. Let $\mathcal{O}$ be the set containing all object types that can be carried between two vertices, i.e., $\mathcal{O} = O \cup \{-1, 0\}$.

Definition 4.2.3. *A triplet (i, j, k), $i \in \mathcal{V}$, $j \in \mathcal{V}$, $j \neq i$, $k \in \mathcal{O}$, is called discardable if the carrying of object type k from vertex i to vertex j cannot be part of an optimal solution. Denote by $\mathcal{N}$ the subset of non-discardable triplets.*

We begin by enumerating some discardable triplets that concern every vertex (Propositions 4.2.6 and 4.2.7), and then analyze particular cases based on the supply and the demand object. All cases are in a way or another implied by Theorem 4.2.1 and the vertex triplication described in Section 4.2.2.

Proposition 4.2.6. *If $i \in V_1$, then the following triplets are discardable: 1) $(j \in \mathcal{V}, i, -1)$; 2) $(i, j \in \mathcal{V} \setminus \{i+n\}, -1)$; 3) $(i+n, j \in \mathcal{V} \setminus \{i+2n\}$; 4) $(i+2n, j \in \mathcal{V}, -1)$; 5) $(i+n, i, k \in \mathcal{O})$; 6) $(i+2n, i, k \in \mathcal{O})$; 7) $(i+2n, i+n, k \in \mathcal{O})$; 8) $(i, i+n, k \in \mathcal{O} \setminus \{-1\})$; 9) $(i+n, i+2n, k \in \mathcal{O} \setminus \{-1\})$; 10) $(i, i+2n, k \in \mathcal{O})$.*

Proof. 1) to 4): by definition object -1 can only be carried between two consecutive copies of the same vertex, i.e., from i to $i+n$ and from $i+n$ to $i+2n$; 5): by definition $i \prec i+n$; 6): by definition $i \prec i+2n$; 7): by definition $i+n \prec i+2n$; 8) and 9): by definition object type -1 is the only available object that can be carried between two consecutive copies of the same vertex; 10): i and $i+2n$ cannot be connected by any arc since $i \prec i+n \prec i+2n$. □

Proposition 4.2.7. *If $i \in V_1$ and $a_i \neq b_i$, then the following triplets are discardable: 1) $(j \in \mathcal{V}, i, a_i)$; 2) $(j \in \mathcal{V}, i+n, a_i)$; 3) $(j \in \mathcal{V}, i+2n, a_i)$; 4) $(i+n, j \in \mathcal{V}, b_i)$; 5) $(i+2n, j \in \mathcal{V}, b_i)$; 6) $(i, j \in \mathcal{V}, b_i)$.*

Proof. 1) to 5): by Theorem 4.2.1 the only objects that can be carried to an original vertex are the null object, the demand object or a dropped object, therefore when $a_i \neq b_i$ there exists an optimal solution in which the supply object is never carried to any of the copies of that original vertex; 6): by Theorem 4.2.1 again, there exists an optimal solution in which the vehicle can only carry from an original vertex the null object, the supply object or an object that has been dropped previously; therefore the demand object never exits any of the copies of that original vertex. □

Now consider the depot and its copies. There are two cases depending on the supply and the demand object (Propositions 4.2.8 and 4.2.9).

Proposition 4.2.8. *If $a_1 = 0$ or $b_1 = 0$ or both, then the following triplets are discardable: 1) $(j \in \mathcal{V}, 1, k \in \mathcal{O} \setminus \{b_1\})$; 2) $(1, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$; 3) $(j \in \mathcal{V} \setminus \{1\}, 1+n, k \in \mathcal{O})$; 4) $(1+n, j \in \mathcal{V} \setminus \{1+2n\}, k \in \mathcal{O})$; 5) $(j \in \mathcal{V}, 1+2n, k \in \mathcal{O} \setminus \{-1\})$; 6) $(1+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{a_1\})$.*

Proof. 1): if a_i or b_i is the null object, then by Theorem4.2.1 no additional deadheading can be incident to the original depot, and there is no drop at the depot. Then there exists an optimal solution in which the original depot is visited only once. Therefore the demand object must be carried to the first copy; 2) and 3): by definition, when the original vertex is visited only once, object -1 must be carried from i to $i+n$; 4) and 5): by definition, when the original vertex is visited only once, object -1 must be carried from $i+n$ to $i+2n$; 6): since there is no null object nor a dropped object entering the original vertex, the only available object at the last copy is the supplied object. □

Proposition 4.2.9. *If $a_1 = b_1$ and $a_1 \neq 0$, or if $a_1 \neq b_1$, then the following triplets are discardable: 1) $(j \in \mathcal{V}, 1, k \in \mathcal{O} \setminus \{0, b_1\})$; 2) $(1, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_1\})$; 3) $(j \in \mathcal{V}, 1+n, k \in \mathcal{O} \setminus \{-1, 0, b_1\})$; 4) $(1+n, j \in \mathcal{V} \setminus \{1+2n\}, k \in \mathcal{O})$; 5) $(j \in \mathcal{V}, 1+2n, k \in \mathcal{O} \setminus \{-1\})$; 6) $(1+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_1\})$.*

Proof. By Theorem 4.2.1 there is no drop at the original depot. Therefore, there exists an optimal solution in which the original depot is visited at most twice, where the two visits case corresponds to an additional pair of deadheadings (incoming and outgoing). Then we deduce that: 1): only a deadheading or an arc carrying the demand object can enter the first copy; 2): object -1 is carried from 1 to $1+n$ when the original vertex is visited only once, otherwise the two possibilities are a deadheading or an arc carrying the supply object; 3): similar to 2) but consider the demand instead of the supply because it is an incoming arc; 4) and 5): since the original vertex is visited at most twice, by definition $1+n$ and $1+2n$ are connected by an arc carrying object -1; 6): since there is no drop at the original depot, the only available objects at the last copy are the null object or the supply object. □

We now enumerate the discardable triplets related to a non-depot vertex. There are four cases (Propositions 4.2.10, 4.2.11, 4.2.12 and 4.2.13).

Proposition 4.2.10. *If $a_i = b_i = 0$, $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(j \in \mathcal{V}, i, 0)$; 2) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0\})$; 3) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{0\})$; 4) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$; 5) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1\})$; 6) $(i+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0\})$.*

Proof. By Theorem 4.2.1 and Proposition 4.2.2 there exists an optimal solution in which an original vertex is either visited twice or not visited at all.
1): if the vehicle carries the null object to the first copy, then there will be two consecutive deadheadings because $a_i = b_i = 0$ (see Proposition 4.2.4); 2): the null object (i.e., the supply object here) is the only available type at the first copy; 3): similar to 2) but consider the supply instead of the demand (they are actually the same here) since it is an incoming arc; 4) and 5): since the original vertex is visited at most twice, by definition the last two copies are connected with an arc carrying object -1; 6): since the original vertex cannot be visited only once, the supply object (i.e., the null object) cannot be carried from the last copy. Only a dropped object, if any, can be available at that copy. □

Proposition 4.2.11. *If $a_i = 0$ or $b_i = 0$ (but not both), $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, a_i\})$; 2) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{-1, b_i\})$; 3) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$; 4) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1\})$.*

Proof. 1): the null object cannot be carried from the first copy because this would imply two consecutive deadheadings since a_i or b_i is 0 (see Proposition 4.2.4); 2): similar to 1) but consider the demand instead of the supply; 3) and 4): since $a_i = 0$ or $b_i = 0$, there are no additional deadheading among the incident arcs. Therefore, by Theorem 4.2.1, the original vertex is visited at most twice, and then by definition, the last two copies are connected by an arc carrying object -1. □

Proposition 4.2.12. *If $a_i = b_i$ and $a_i \neq 0$, $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_i\})$; 2) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{0, b_i\})$; 3) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_i\})$; 4) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$.*

Proof. 1): by Theorem 4.2.1, if there is a drop at a vertex, then the dropped object is reloaded during the last exit (i.e., at the third copy). Therefore the only available objects at the first copy are the null object and the supply object; 2): similar to 1) but consider the demand instead of the supply since it is an incoming arc; 3) similar to 1), the only available object at $i+n$ are 0 or a_i since a dropped object, if any, can only be

reloaded at $i+2n$; 4): by Theorem 4.2.1 and by definition of the triplication, a drop can only occur at the first copy. Then any arc entering the last copy must carry -1, 0 or the demand object. □

Proposition 4.2.13. *If $a_i \neq b_i$ and $a_i \neq 0$, $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_i\})$; 2) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_i\})$; 3) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$; 4) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$.*

Proof. 1) and 2): by Theorem 4.2.1 a dropped object is reloaded during the last exit, i.e., at the last copy. Therefore only -1, 0 or a_i can be carried from i; 3): similar to 1) but consider the demand instead of the supply object; 4): similar to 3) for the third copy. □

4.3 Mathematical model

This section presents a mixed integer linear program for the preemptive SP. The allowable triplet configurations described in Section 4.2 are handled through the variables and the constraints. To each triplet $(i, j, k) \in \mathcal{N}$, is associated a binary variable x_{ij}^k equal to 1 if and only if an object of type k is carried from i to j, and to each vertex $i \in \mathcal{V}$ is associated a real variable u_i indicating the position of i in the route ($u_1 = 0$ and $1 \leq u_i \leq 3n-1$, $i = 2, \ldots, 3n$). The u_i variables are similar to those introduced by Miller, Tucker and Zemlin (MTZ) in their TSP formulation ([43]). The possible optimal configurations for a vertex depend on its supply and its demand. The constraints relative to each case are presented separately. The idea is to restrict the search for a solution to the subspace of solutions that satisfy optimal structural properties. Every variable representing a discardable triplet must be interpreted as zero.

4.3.1 Depot vertex with $a_1 = 0$ or $b_1 = 0$

By Theorem 4.2.1, objects cannot be dropped at the depot, and for any original vertex the demand (supply) object is carried only once to (from) that vertex. Since $a_1 = 0$ or

$b_1 = 0$ and because of Proposition 4.2.4, the depot is visited only once (constraints 4.1 and 4.4). Therefore after triplication the three copies are connected by two arcs carrying object -1 as shown in Figure 4.8 (constraints 4.2 and 4.3). The corresponding constraints are

$$\sum_{j \in \mathcal{V}} x_{j1}^{b_1} = 1 \tag{4.1}$$

$$x_{1,1+n}^{-1} = 1 \tag{4.2}$$

$$x_{1+n,1+2n}^{-1} = 1 \tag{4.3}$$

$$\sum_{j \in \mathcal{V}} x_{i+2n,j}^{a_1} = 1. \tag{4.4}$$

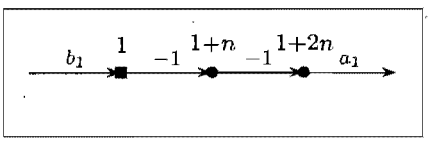


FIG. 4.8 – Configuration for the depot with $a_1 = 0$ or $b_1 = 0$

4.3.2 Depot vertex with $a_1 \neq 0$ and $b_1 \neq 0$

Since neither a_1 nor b_1 is the null object, a pair of deadheadings (incoming and outgoing) can possibly be used among the arcs incident to the three copies of the depot. By Proposition 4.2.4, there exist only two ways of visiting the original depot twice. This leads to the three possible configurations depicted in Figure 4.9. Constraints 4.5 and 4.6 ensure that the demand and the supply are satisfied. Since the original depot is visited at most twice, its second and third copies are connected by an arc carrying object -1 (constraint 4.7). Constraint 4.8 takes care of the conservation of the null object, which must be reloaded at the third copy if it has been dropped at the first copy (this is similar to the drop of a real object). The corresponding constraints are

$$\sum_{j \in \mathcal{V}} \left(x_{j1}^{b_1} + x_{j,1+n}^{b_1} \right) = 1 \tag{4.5}$$

$$\sum_{j \in \mathcal{V}} \left(x_{1j}^{a_1} + x_{1+2n,j}^{a_1} \right) = 1 \tag{4.6}$$

$$x_{1+n,1+2n}^{-1} = 1 \tag{4.7}$$

$$\sum_{j \in \mathcal{V}} \left(x_{j1}^{0} - x_{1+2n,j}^{0} \right) = 0. \tag{4.8}$$

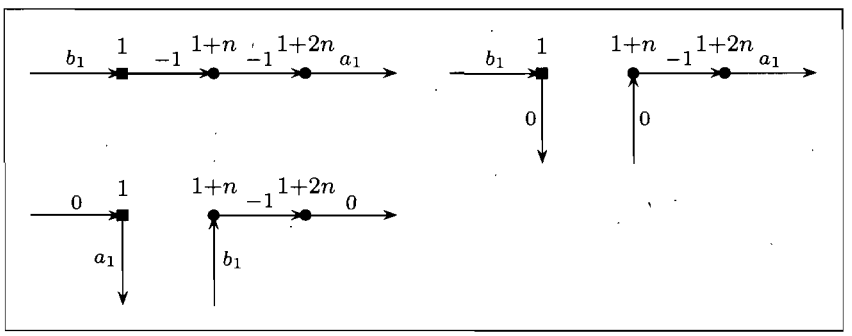


FIG. 4.9 – Configurations for the depot with $a_1 \neq 0$ and $b_1 \neq 0$

4.3.3 Non-depot vertex with $a_i = b_i = 0$

Since $a_i = b_i$, the original vertex is a transshipment vertex and the vehicle can possibly skip it during its tour (the demand is already satisfied by the supply). Since both a_i and b_i are the null object, no additional deadheading can be incident to the three copies of the vertex, except the two arcs that carry the supply and the demand themselves, which can be carried from the first copy and to the second copy, respectively. Furthermore, by Theorem 4.2.1, if there is a drop at vertex i, it occurs during the first entry and the reloading takes place during the last exit, yielding constraints 4.9 and 4.11. Since the original vertex is visited at most twice, the second and the third copies must be connected by an arc carrying object -1, which only occurs when the second copy has received its demand (constraints 4.10). Figure 4.10 represents the two possible configurations. Note that the vehicle cannot visit the original vertex only once because

it is a transshipment vertex (see Proposition 4.2.2). We consider the following set of constraints: $\forall i \in \{j \in V_1 \setminus \{1\} \mid a_j = b_j = 0\}$,

$$\sum_{k \in \mathcal{O}} \sum_{j \in \mathcal{V}} x_{ji}^{k} - \sum_{j \in \mathcal{V}} x_{ij}^{a_i} = 0 \tag{4.9}$$

$$\sum_{j \in \mathcal{V}} x_{j,i+n}^{b_i} - x_{i+n,i+2n}^{-1} = 0 \tag{4.10}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ji}^{k} - x_{i+2n,j}^{k} \right) = 0 \qquad (k = 1, \ldots, m). \tag{4.11}$$

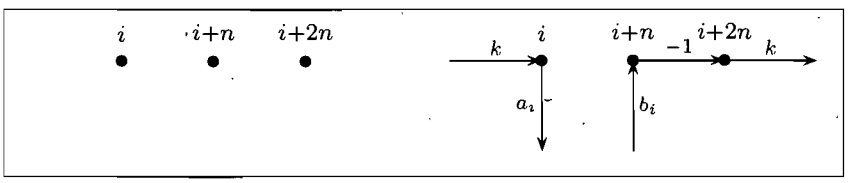


FIG. 4.10 – Configurations for a non-depot vertex with $a_i = b_i = 0$

4.3.4 Non-depot vertex with $a_i \neq b_i$, $a_i = 0$ or $b_i = 0$

The supply of vertex i differs from its demand, so the vehicle must necessarily carry b_i (a_i) to (from) the original vertex, which is visited at most twice because there is no additional deadheading among the incident arcs (since a_i or b_i is the null object). The demand can be carried to either the first copy (when the original vertex is visited only once) or to the second copy (when there is a drop) (constraints 4.12), and the supply can exit the first copy or the third copy (constraints 4.13). If the vehicle carries the demand to the first copy, then the original vertex is visited only once and the first two copies must be connected by an arc carrying object -1 (constraints 4.14). Furthermore, since the original vertex cannot be visited more than twice, then the last two copies are also connected by an arc carrying object -1 (constraints 4.15). By Theorem 4.2.1, if there is a drop it must occur during the first entry and the reloading occurs during the last exit (constraints 4.16). The two possible configurations are shown in Figure 4.11. We consider the following set of constraints: $\forall i \in \{j \in V_1 \setminus \{1\} \mid a_j = 0 \text{ or } b_j = 0\}$,

$$\sum_{j\in\mathcal{V}}\left(x_{ji}^{b_i}+x_{j,i+n}^{b_i}\right) = 1 \tag{4.12}$$

$$\sum_{j\in\mathcal{V}}\left(x_{ij}^{a_i}+x_{i+2n,j}^{a_i}\right) = 1 \tag{4.13}$$

$$\sum_{j\in\mathcal{V}} x_{ji}^{b_i} - x_{i,i+n}^{-1} = 0 \tag{4.14}$$

$$x_{i+n,i+2n}^{-1} = 1 \tag{4.15}$$

$$\sum_{j\in\mathcal{V}}\left(x_{ji}^{k}-x_{i+2n,j}^{k}\right) = 0 \qquad (k=1,\ldots,m). \tag{4.16}$$

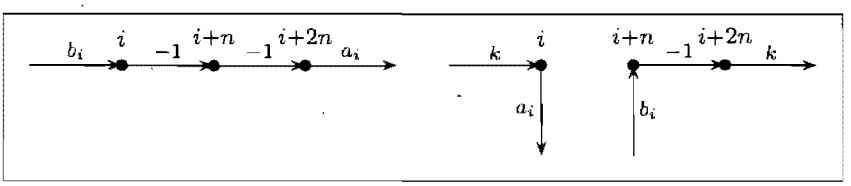


FIG. 4.11 – Configurations for a non-depot vertex with $a_i = 0$ or $b_i = 0$

4.3.5 Non-depot vertex with $a_i = b_i$ and $a_i \neq 0$

Here we handle a more complicated case because the vehicle may visit the original vertex three times due to the fact that the supply and demand object differ from the null object. This situation induces additional deadheading. Also note that since $a_i = b_i$ (i.e., i is a transshipment vertex), the original vertex is not necessarily visited in the solution. When the vehicle visits the original vertex, then there exist three alternatives if it visits the original vertex twice, and only two if it visits the original vertex three times (see Figure 4.12). Again these alternatives come from the fact that there exists an optimal solution that does not contain two consecutive arcs carrying the same object type (Proposition 4.2.4). Constraints 4.17 ensure that any drop occurs at the first copy of the vertex, and the reloading occurs at third copy (Theorem 4.2.1). Constraints 4.18, 4.19 and 4.20 ensure that the supply and the demand are satisfied and take care of the conservation of additional deadheading. We consider the following constraints: $\forall i \in \left\{j \in V_1 \setminus \{1\} \,\middle|\, a_j = b_j, a_j \neq 0\right\}$,

$$\sum_{j\in\mathcal{V}}\left(x^k_{ji} - x^k_{i+2n,j}\right) = 0 \qquad (k = 0,\ldots,m) \tag{4.17}$$

$$\sum_{j\in\mathcal{V}}\left(x^0_{ij} - x^0_{j,i+n}\right) = 0 \tag{4.18}$$

$$\sum_{j\in\mathcal{V}}\left(x^{a_i}_{ij} - x^{b_i}_{j,i+n}\right) = 0 \tag{4.19}$$

$$\sum_{j\in\mathcal{V}}\left(x^{a_i}_{i+n,j} - x^{b_i}_{j,i+2n}\right) = 0. \tag{4.20}$$

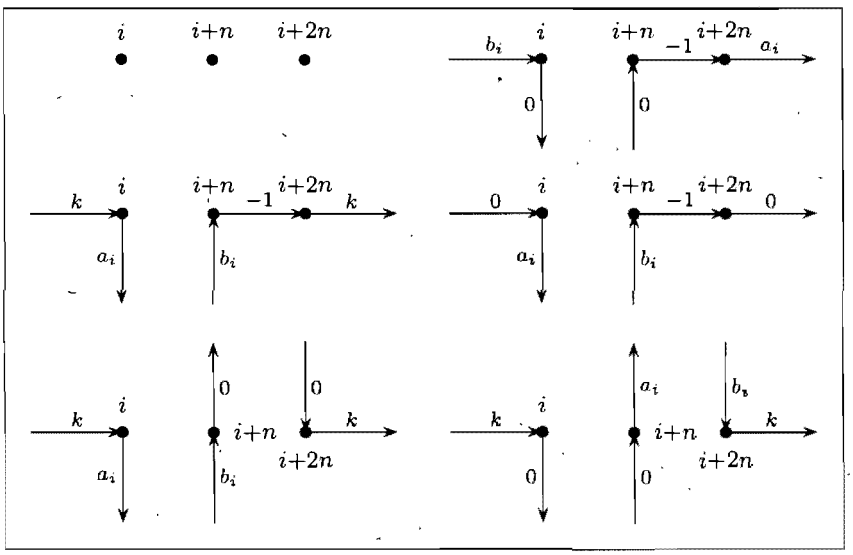


FIG. 4.12 – Configurations for a non-depot vertex with $a_i = b_i$ and $a_i \neq 0$

4.3.6 Non-depot vertex with $a_i \neq b_i$ and $a_i \neq 0, b_i \neq 0$

This case is the most common: it corresponds to the situation where the demand is different from the supply, and both are non-zero. There exists an additional possible configuration with respect to the case described in Section 4.3.5 because $a_i \neq b_i$. Indeed, when the original vertex is visited three times, there may exist a route in which the vehicle carries the demand to the second copy and immediately loads the supply. This could not happen in the case of Section 4.3.5 because both the demand and the supply were identical. As a consequence the demand (supply) object can be

carried to (from) any of the three copies. This yields the seven possible optimal configurations represented in Figure 4.13. We consider the following set of constraints: $\forall i \in \{j \in V_1 \setminus \{1\} \mid a_j \neq b_j, a_j \neq 0, b_j \neq 0\}$,

$$\sum_{j\in\mathcal{V}} \left(x_{ji}^{b_i} + x_{j,i+n}^{b_i} + x_{j,i+2n}^{b_i}\right) = 1 \tag{4.21}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ij}^{a_i} + x_{i+n,j}^{a_i} + x_{i+2n,j}^{a_i}\right) = 1 \tag{4.22}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ji}^{b_i} - x_{i+2n,j}^{a_i}\right) = 0 \tag{4.23}$$

$$\sum_{j\in\mathcal{V}} x_{ji}^{b_i} - x_{i+n,i+2n}^{-1} \leq 0 \tag{4.24}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ji}^{b_i} - x_{ij}^{0}\right) - x_{i,i+n}^{-1} \leq 0 \tag{4.25}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ij}^{a_i} - x_{j,i+n}^{b_i}\right) \leq 0 \tag{4.26}$$

$$\sum_{j\in\mathcal{V}} x_{j,i+n}^{b_i} - \sum_{j\in\mathcal{V}} \left(x_{ij}^{a_i} + x_{i+n,j}^{a_i}\right) \leq 0 \tag{4.27}$$

$$\sum_{j\in\mathcal{V}} x_{i+n,j}^{a_i} - \sum_{j\in\mathcal{V}} \left(x_{j,i+2n}^{0} + x_{j,i+2n}^{b_i}\right) \leq 0 \tag{4.28}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ji}^{k} - x_{i+2n,j}^{k}\right) \leq 0 \quad (k = 0, \ldots, m). \tag{4.29}$$

Constraints 4.21 and 4.22 ensure that the demand is satisfied and the supply is carried to another vertex. By Theorem 4.2.1, if there is a drop, it occurs at vertex $i \in V_1$ and the dropped object is reloaded at vertex $i + 2n$. Therefore, if the vehicle carries b_i to vertex i, then there is no drop at i and the only available object at vertex $i+2n$ is the supply object (constraints 4.23). If there is no drop, then the original vertex must be visited at most twice (Theorem 4.2.1) and, by definition, vertices $i + n$ and $i+2n$ are connected by an arc carrying -1 (constraints 4.24). If the demand b_i is carried to vertex i, then there are two ways of exiting i: carrying the null object or carrying object -1 to vertex $i + n$ (constraints 4.25). When a_i is carried from vertex i, then b_i must be carried to vertex $i+n$, otherwise there would be two consecutive arcs carrying the same object type which is non-optimal by Proposition 4.2.4 (constraints 4.26).

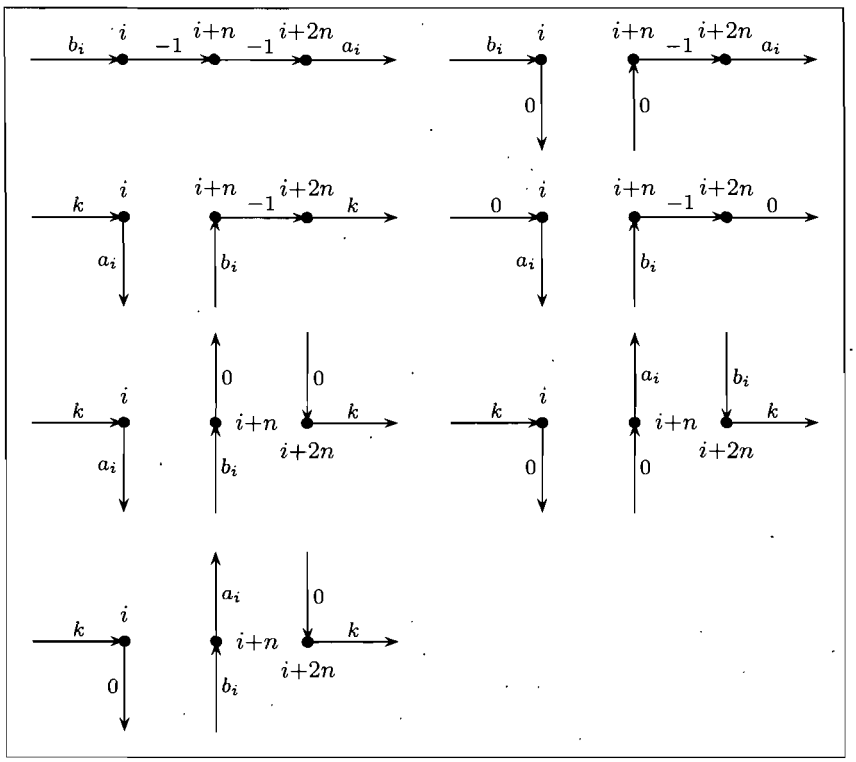


FIG. 4.13 – Configurations for a non-depot vertex with $a_i \neq b_i$ and $a_i \neq 0, b_i \neq 0$

Constraints 4.27 are similar to 4.26 but apply to vertex $i+n$. If the vehicle carries b_i to vertex $i+n$, then in order to avoid two consecutive deadheadings, object a_i must exit from i or $i+n$. If the vehicle exits from $i+n$ with object a_i, then this implies that the original vertex is visited three times. Therefore an object has been dropped at the first copy and must be reloaded at the last copy. Again to avoid two consecutive arcs carrying the same object type (see Proposition 4.2.4), the vehicle must carry either 0 or b_i to vertex $i+2n$ (constraints 4.28). Finally constraints 4.29 ensure that any object dropped at vertex i is reloaded at vertex $i+2n$ (Theorem 4.2.1).

4.3.7 Degree constraints

The three copies of the depot and every non-transshipment vertex have an in-degree and an out-degree equal to one. Therefore we consider the following degree constraints:
$\forall i \in \mathcal{V} \setminus \{j \in \mathcal{V} \setminus \{1, 1+n, 1+2n\} \,|\, a_j \neq b_j\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} x_{ji}^k = 1 \tag{4.30}$$

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} x_{ij}^k = 1. \tag{4.31}$$

For transshipment vertices (which are not necessarily part of an optimal solution) we consider the general conservation constraints which ensure that the in-degree is equal to the out-degree (this is also true for non-transshipment vertices but it is implied by constraints 4.30 and 4.31): $\forall i \in \{j \in \mathcal{V} \,|\, a_j = b_j\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x_{ji}^k - x_{ij}^k\right) = 0. \tag{4.32}$$

If one copy of a transshipment vertex is visited, then the other copies must be visited as well (again this is also true for non-transshipment vertices but it is implied by constraints 4.30 and 4.31). This can be expressed by the following constraints: $\forall i \in \{j \in V_1 \,|\, a_j = b_j\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x_{ji}^k - x_{j,i+n}^k\right) = 0 \tag{4.33}$$

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x_{j,i+n}^k - x_{j,i+2n}^k\right) = 0. \tag{4.34}$$

4.3.8 Subtour elimination constraints

Standard subtour elimination constraints (SEC) for the directed TSP state that in any optimal solution the vehicle must leave (and equivalently enter) every proper subset of vertices at least once. As we have seen, in the preeemptive SP the non-depot vertices whose supply and demand are the same may or may not be part of an optimal solution. Therefore standard SEC must be slightly modified. Actually, subtour elimination con-

straints 4.35 are the same as in the directed TSP, but the subset of vertices to which they apply must satisfy additional requirements.

Definition 4.3.1. *A subset $U \subset \mathcal{V}$ is called SEC-compatible if it is valid to impose an SEC on U. Denote by $\mathcal{U}$ the family of SEC-compatible subsets.*

Proposition 4.3.1. *$U \in \mathcal{U}$ if and only if it satisfies one of the two conditions: 1) at least one copy of the depot is in U, and at least one copy of the depot is not in U or there exists at least one non-transshipment vertex not in U; 2) at least one copy of the depot is not in U, and at least one copy of the depot is in U or there exists at least one non-transshipment vertex in U.*

Proof. $\Rightarrow$ Let $U \subset \mathcal{V}$. The vertices that are visited in all solutions are the three copies of the depot and the non-transshipment vertices. By Definition 4.3.1, it is feasible to impose an SEC on U, which means that any feasible circuit must exit U at least once. This implies that the vehicle must visit at least one vertex in U and at least another vertex not in U. There are two cases. In the first one $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \{1, 1+n, 1+2n\}$ not in U $(i \neq j)$. In the second one $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \mathcal{V} \setminus \{1, 1+n, 1+2n\}$ such that $a_j \neq b_j$. This gives us the first condition of the proposition. The second condition follows by considering the complement of U.
$\Leftarrow$ Suppose $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \{1, 1+n, 1+2n\}$ not in U. Since any feasible solution visits the three copies of the depot, this implies that the vehicle must exit U at least once, which corresponds to an SEC on U. Similarly, suppose $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \mathcal{V} \setminus \{1, 1+n, 1+2n\}$ such that $a_j \neq b_j$. Since the three copies of the depot are visited by any solution and every non-transshipment vertex must also be visited (in order to satisfy its demand and its supply), the vehicle must exit U at least once, which corresponds to an SEC on U. The second condition of the proposition follows by considering the complement of U. □

Subtour elimination constraints for the preemptive SP can be written as follows: $\forall U \in \mathcal{U}$,

$$\sum_{k \in \mathcal{O}} \sum_{i \in U} \sum_{j \notin U} x_{ij}^k \quad \geq \quad 1. \tag{4.35}$$

4.3.9 u-precedence constraints

The u_i variable indicates the position of vertex i in the tour. Since $i \prec i+n$ and $i+n \prec i+2n$, every feasible solution must satisfy the following constraints: $\forall\, i \in V_1$,

$$u_{i+n} - u_i \geq 1 \tag{4.36}$$

$$u_{i+2n} - u_{i+n} \geq 1. \tag{4.37}$$

4.3.10 MTZ constraints

Miller et al. [43] introduced the so-called MTZ constraints for the TSP (on n vertices):

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \qquad (i,\, j = 2, \ldots, n).$$

Desrochers and Laporte [23] have later shown that these constraints can be lifted as follows:

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 \qquad (i,\, j = 2, \ldots, n).$$

In our context we must consider $3n$ vertices and the object types. The lifted MTZ constraints become: $\forall\, i, j \in \mathcal{V} \setminus \{1\}$,

$$u_i - u_j + (3n-1)\sum_{k \in \mathcal{O}} x_{ij}^k + (3n-3)\sum_{k \in \mathcal{O}} x_{ji}^k \leq 3n-2. \tag{4.38}$$

These constraints mean that if the vehicle travels from i to j, then u_j should be greater or equal than $u_i + 1$. They are efficient only if they are combined with the u-precedence constraints: they link the u_i to the x_{ij}^k , but not necessarily sequence the u_i variables in a feasible way at an integer solution. On the other hand, the u-precedence constraints impose precedence relationships but are not sufficient to guarantee a feasible vehicle route. However, the presence of these two sets of constraints in the model ensures that any integer solution satisfies all precedence relationships.

4.3.11 Comb inequalities

Comb inequalities were first identified by Chvátal [19] and then generalized by Grötschel and Padberg [33]. A comb is defined by a subset $H \subset V$, called the *handle*, and an odd number $t \geq 3$ of *teeth* $T_i \subset V$ $(i = 1, \ldots, t)$ such that (see Figure 4.14):

$$\begin{aligned} H \cap T_i &\neq \emptyset\ , \quad (i = 1, \ldots, t) \\ T_i \setminus H &\neq \emptyset\ , \quad (i = 1, \ldots, t) \\ T_i \cap T_j &\neq \emptyset\ , \quad (1 \leq i < j \leq t). \end{aligned}$$

For the TSP the comb inequality is expressed as:

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq 3t + 1,$$

where $\delta(S)$ is the set of edges having only one endpoint in S, and $x(S)$ represents the sum of the values of the edges having their two endpoints in S.

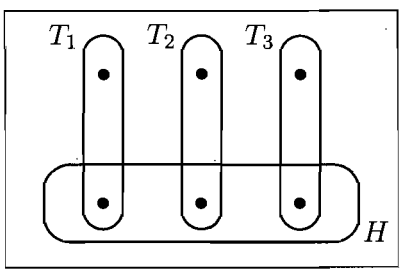


FIG. 4.14 – Minimal comb configuration

In the preemptive SP the graph is directed and in a solution each arc is associated with an object type. Simply replacing each arc (i, j) by an edge $e = (i, j)$ with value $x_e = \sum_{k=0}^{m}(x_{ij}^k + x_{ji}^k)$ is not sufficient to define a valid comb inequality. It is necessary to also take into account the fact that some vertices are not required in the solution (i.e., transshipment vertices). The subsets H and T_i must satisfy an additional requirement which is introduced in the next definition.

Definition 4.3.2. *A subset $H \subset V$ and an odd number $t \geq 3$ of subsets $T_i \subset V$ $(i = 1, \ldots, t)$ such that 1) $H \cap T_i \neq \emptyset$, $(i = 1, \ldots, t)$, 2) $T_i \setminus H \neq \emptyset$, $(i = 1, \ldots, t)$ and 3) $T_i \cap T_j \neq \emptyset$, $(1 \leq i < j \leq t)$, is said to be comb-compatible if it is valid to impose a comb inequality from the handle H and the teeth T_i.*

Proposition 4.3.2. *A comb (H, T_i) is comb-compatible if the following two conditions are satisfied for each T_i: 1) at least one copy of the depot is in $T_i \setminus H$ or there exists at least one non-transshipment vertex in $T_i \setminus H$; 2) at least one copy of the depot is in $T_i \cap H$ or there exists at least one non-transshipment vertex in $T_i \cap H$.*

Proof. The proof is similar to that of Proposition 4.3.1. If a vertex has the same demand and supply (i.e., a transshipment vertex) and is not a copy of the depot, the vehicle can possibly skip it in an optimal solution. Such a vertex is somehow unconstrained since its supply can satisfy its own demand. On the other hand each copy of the depot and the vertices i with $a_i \neq b_i$ must necessarily be visited by any feasible solution. Since the minimal comb configuration (see Figure 4.14) consists of exactly one vertex in each $T_i \setminus H$ and $T_i \cap H$, having one copy of the depot or a vertex i with $a_i \neq b_i$ in each of these subsets forces the vehicle to cross each border an appropriate number of times to satisfy the standard comb inequality. □

4.3.12 π-inequalities

Dropping an object at a vertex induces precedence relationships in the vehicle route. Therefore, cuts developed for other problems such as the SOP and the *asymmetric TSP with precedence constraints* (PCATSP) (see Ascheuer et al. [9], Balas et al. [11]) can be used to strengthen our formulation.

Definition 4.3.3. *For all $j \in \mathcal{V} \setminus \{1\}$, let $\pi(j)$ be the subset of vertices, excluding the depot, that must be visited before j in any feasible solution, i.e., $\pi(j) = \{i \in \mathcal{V} \setminus \{1\} \mid i \prec j\}$. The elements of $\pi(j)$ are called the predecessors of j.*

Definition 4.3.4. *Given $S \subset \mathcal{V}$, $\pi(S)$ is the set of its predecessors, i.e., $\pi(S) = \{i \in \pi(j) \mid j \in S\}$.*

The π-inequalities were introduced by Balas et al. [11] and can be expressed as follows for the PCATSP defined on a graph $G = (V, A)$ with vertex 1 as a depot: $\forall S \subset V \setminus \{1\}$,

$$x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \quad \geq \quad 1.$$

These constraints mean that in any feasible solution there must be a path from any vertex $j \in S$ to vertex 1 that does not pass through $\pi(j)$. For the preemptive SP, due to the presence of transshipment vertices, an additional requirement must be satisfied by the subsets S to which the constraints apply.

Definition 4.3.5. *A subset $S \subset \mathcal{V} \setminus \{1\}$ is called π-compatible if it is valid to impose a π-inequality on S. Denote by Π the family of π-compatible subsets.*

Proposition 4.3.3. *A subset $S \subset \mathcal{V} \setminus \{1\} \in \Pi$ if there exists at least one non-transshipment vertex in $S \setminus \pi(S)$.*

Proof. Let $S \subset \mathcal{V} \setminus \{1\}$. Suppose that $S \in \Pi$ and $a_j = b_j$, $\forall j \in S \setminus \pi(S)$. Since $S \setminus \pi(S)$ contains only transshipment vertices, the inequality $x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \geq 1$ is clearly not valid because we cannot force the vehicle to exit $S \setminus \pi(S)$ since the demand of each vertex in $S \setminus \pi(S)$ is already satisfied by its own supply. This contradicts our assumption that $S \in \Pi$. □

The π-inequalities for the preemptive SP become: $\forall\, S \in \Pi$,

$$x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \quad \geq \quad 1. \tag{4.39}$$

4.3.13 σ-inequalities

The σ-inequalities are similar to the π-inequalities, the role of the predecessors is simply interchanged with the successors.

Definition 4.3.6. *For all $i \in \mathcal{V} \setminus \{1\}$, let $\sigma(i)$ be the subset of vertices, excluding the depot, that must be visited after i in any feasible solution, i.e., $\sigma(i) = \{j \in \mathcal{V} \setminus \{1\} \mid j \succ i\}$. The elements of $\sigma(i)$ are called the successors of i.*

Definition 4.3.7. *Given $S \subset \mathcal{V}$, $\sigma(S)$ is the set of its successors, i.e., $\sigma(S) = \{j \in \sigma(i) \mid i \in S\}$.*

The σ-inequalities were introduced by Balas et al. [11] and can be expressed as follows for the PCATSP defined on a graph $G = (V, A)$ with vertex 1 as a depot:

$\forall S \subset V \setminus \{1\}$,

$$x\left(\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)\right) \quad \geq \quad 1.$$

These constraints mean that in any feasible solution there must be a path from vertex 1 to any vertex $i \in S$ that does not pass through $\sigma(i)$. For the preemptive SP, due to the presence of transshipment vertices, an additional requirement must be satisfied by the subsets S to which the constraints apply.

Definition 4.3.8. *A subset $S \subset \mathcal{V} \setminus \{1\}$ is called σ-compatible if it is valid to impose a σ-inequality on S. Denote by Σ the family of σ-compatible subsets.*

Proposition 4.3.4. *A subset $S \subset \mathcal{V} \setminus \{1\} \in \Sigma$ if there exists at least one non-transshipment vertex in $\bar{S} \setminus \sigma(S)$.*

Proof. The proof is similar by symmetry to that of Proposition 4.3.3. □

The σ-inequalities for the preemptive SP become: $\forall S \in \Sigma$,

$$x\left(\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)\right) \quad \geq \quad 1. \tag{4.40}$$

Note that in our problem the π- and σ-inequalities do not consider transshipment vertices because these are not necessarily part of an optimal solution. However, there also exist precedence relationships for transshipment vertices; if the vehicle visits such vertices these precedence relationships must be satisfied as well. That is the reason why we have introduced the MTZ constraints 4.38 (combined with the u-precedence constraints 4.36 and 4.37). Although they are much weaker than the π- and σ-inequalities because they may be inefficient at fractional solution, they apply to all vertices, even those that are not visited in the solution.

4.3.14 Generalized order constraints

The generalized order constraints (GOC) were introduced by Ruland [46] for the *Pickup and Delivery Problem*. Similar constraints were proposed by Balas et al. [11] under

the name of *precedence cycle breaking inequalities*. These constraints are defined as follows. Let $S_1, \ldots, S_t \subset \mathcal{V} \setminus \{1\}$ be $t \geq 2$ disjoint subsets such that $S_i \cap \pi(S_{i+1}) \neq \emptyset$, $\forall\, i = 1, \ldots, t$, where $S_{t+1} = S_1$. The GOC is given by the inequality

$$\sum_{i=1}^{t} x(S_i) \quad \leq \quad \sum_{i=1}^{t} |S_i| - t - 1.$$

The GOC can be viewed as a tightened version of the SEC. Cordeau [21] has shown that they can be lifted by adding extra arcs to their left-hand side. Figure 4.15 illustrates a simple GOC configuration with $t = 3$ and $|S_i| = 2$ $(i = 1, 2, 3)$.

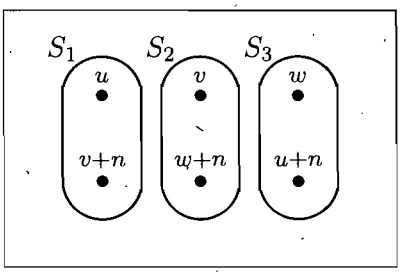


FIG. 4.15 – Example of GOC configuration

4.3.15 Formulation

Our model for the preemptive SP is as follows:

minimize	$\sum_{(i,j,k)\in\mathcal{N}} c_{ij} x_{ij}^{k}$	
subject to	constraints (4.1)–(4.4)	if a_1 or $b_1 = 0$
	constraints (4.5)–(4.8)	if $a_1 \neq b_1$ and $a_1 \neq 0$
	constraints (4.9)–(4.11)	for all $i \in V_1 \setminus \{1\} \mid a_i = b_i = 0$
	constraints (4.12)–(4.16)	for all $i \in V_1 \setminus \{1\} \mid a_i$ or $b_i = 0$
	constraints (4.17)–(4.20)	for all $i \in V_1 \setminus \{1\} \mid a_i = b_i$ and $a_i \neq 0$
	constraints (4.21)–(4.29)	for all $i \in V_1 \setminus \{1\} \mid a_i \neq b_i$, $a_i \neq 0$ and $b_i \neq 0$
	constraints (4.30)–(4.31)	for all $i \in \mathcal{V} \mid a_i \neq b_i$

constraints (4.32)–(4.34)	for all $i \in \mathcal{V}$
constraint (4.35)	for all $U \in \mathcal{U}$
constraints (4.36)–(4.37)	for all $i \in V_1$
constraint (4.38)	for all $i, j \in \mathcal{V} \setminus \{1\}$
$u_1 = 0$	
$1 \leq u_i \leq 3n - 1$	for all $i \in \mathcal{V} \setminus \{1\}$
$x_{ij}^k \in \{0, 1\}$	for all $(i, j, k) \in \mathcal{N}$.

4.4 Branch-and-cut algorithm

We have developed a branch-and-cut algorithm for the preemptive SP. Initially the subtour elimination constraints 4.35, the MTZ constraints 4.38 and the integrality constraints are relaxed. We denote the current linear program by LP.

Whenever the relaxation is solved, an attempt is made to detect some violated inequalities of a certain type. If some are detected, they are added to the current relaxation which is solved again. The process continues until no more violated inequalities can be identified. At this point, if the optimal LP solution satisfies the integrality constraints and all precedence relationships are satisfied, then the current solution becomes the new incumbent. Otherwise, branching is performed (see the following pseudo-code of *NodeTreatment*).

4.4.1 Separation of inequalities

We now explain the various separation procedures and the branching rules used in our algorithm.

Procedure 4.4.1 *NodeTreatment*
Input: LP

```
1: solve LP
2: if violated subtour found then
3:         separate
4:         goto 1
5: else if violated combs found then
6:         separate
7:         goto 1
8: else if violated π-inequalities found then
9:         separate
10:        goto 1
11: else if violated σ-inequalities found then
12:        separate
13:        goto 1
14: else if x* is integer then
15:        if violated precedence found then
16:                separate by adding lifted MTZ for each 1-arc
17:                goto 1
18:        else
19:                incumbent ← x*
20:        end if
21: else
22:        branch
23: end if
```

Exact separation of subtour elimination constraints

To separate the SEC we use the Hao-Orlin algorithm ([34]) to compute a global minimum cut in the support graph of the current solution. If the value of the minimum cut is strictly less than 1, then we check for SEC-compatibility (see Proposition 4.3.1). When the cutset is SEC-compatible, the corresponding constraint is added to the LP

which is solved again.

Heuristic separation of comb inequalities

Comb inequalities are normally applied to the symmetric TSP. However, adjustments must be made for the preemptive SP because the underlying graph G is directed. A solution is determined by a list of arcs, each associated with an object type. There is no known polynomial algorithm to separate general combs and this problem is most likely NP-hard. Therefore some heuristic method must be considered. In our implementation we use two different heuristics that both take as input a list of edges with their corresponding value (obtained when the LP is solved). We first transform each arc (i, j) into an edge $e = (i, j)$ with value $x_e = \sum_{k \in \mathcal{O}} (x_{ij}^k + x_{ji}^k)$ and apply a comb violation heuristic: if the search is successful, then we check for comb-compatibility (see Proposition 4.3.2). When comb-compatibility is satisfied we add the corresponding comb inequality to the LP. The first heuristic comes from the publicly available package CVRPSEP developed by Lysgaard (see [41]) for the *Capacitated Vehicle Routing Problem* ([40, 42]), and the second one is our implementation of the heuristic described by Naddef and Thienel [45]. For the latter the idea is to detect a candidate for the handle first and then try to find an appropriate set of teeth by means of a growing sets technique. The two heuristics are executed subsequently and only distinct cuts are kept.

Exact separation of π-inequalities

To detect a violated π-inequality, we apply the exact procedure described in [11]. For every vertex $i \in V_1 \setminus \{1\}$ with $a_i \neq b_i$, we construct a new graph G'_{i+n} from the current support graph, having the same vertex set and the same arc set, except that we remove $\pi(i+n)$ (i.e., vertex i), and its incident arcs. Next, we try to send one unit of flow in G'_{i+n} from $i+n$ to the depot (see Figure 4.16). We actually compute a minimum s-t cut in G'_{i+n}, where $s = i+n$ and $t = 1$. If the value of the minimum cut is strictly less than 1, then a violated π-inequality has been detected and the identified cutset is used to build the constraint. We proceed in a similar way for each vertex $i+n \in V_2 \setminus \{1+n\}$

with $a_{i+n} \neq b_{i+n}$, constructing G'_{i+2n} (this time we delete i, $i+n$ and their incident arcs) and determining the value of a maximum flow from $i+2n$ to the depot. All violated inequalities are stored and are added to the LP at the end of the procedure.

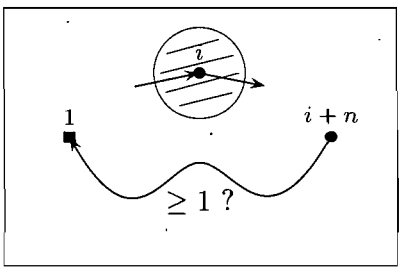


FIG. 4.16 – Illustration of the π-inequality separation procedure

Exact separation of σ-inequalities

The detection procedure for σ-inequality violations is similar to the one applied to π-inequalities. For each vertex $i \in V_1 \setminus \{1\}$ with $a_i \neq b_i$, we first construct a graph G'_i from the current support graph with the same vertex set and the same arc set except that we remove $i+n$, $i+2n$ and their incident arcs. Then we determine a minimum s-t cut in G'_i, where $s = 1$ and $t = i$. If the value of the minimum cut is strictly less than 1, we have found a violated σ-inequality. The same process is executed for each vertex $i+n \in V_2 \setminus \{1+n\}$ with $a_{i+n} \neq b_{i+n}$, creating the graph G'_{i+n} (by copying the support graph and deleting $i+n$ and its incident arcs), computing a minimum s-t cut, where $s = 1$ and $t = i+n$. Again if the value of the minimum cut is strictly less than 1, a violated σ-inequality has been identified and is stored. Figure 4.17 illustrates the main idea of the procedure.

For the calculation of the successive minimum s-t cuts we use the routine *CC-cut_mincut_st* from Concorde (see [5]). It is a very fast implementation of the push-relabel flow algorithm described by Goldberg and Tarjan [32].

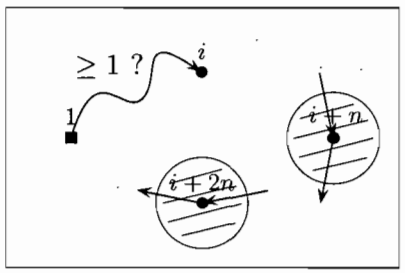


FIG. 4.17 – Illustration of the σ-inequality separation procedure

Precedence for transshipment vertices

Because we decided not to include the MTZ constraints in the LP and the π- and σ-inequalities do not consider transshipment vertices, there may exist violated precedence relationships among these vertices. Therefore at the end of the separation phase, if the solution is on integer circuit we check whether the precedence conditions are satisfied. For this, we follow the route starting at the depot and as soon as a violated precedence relationship is detected, for instance a vertex $i+n \in V_2$ being visited before $i \in V_1$, the process stops and a lifted MTZ constraint is added for each arc having value 1 in the current solution (see the pseudo-code of *NodeTreatment*).

Generalized order constraints

Separating GOC constraints with an exact method can be very time consuming even for the simplest GOC with $t = 2$ and $|S_1| = |S_2| = 2$. For this case Ruland [46] (see also Ruland and Rodin [47]) has proposed an exact separation procedure for the PDP that requires solving $O(n^2)$ maximum flow problems. In our context there are $3n$ vertices and two different precedence relationships to consider ($i \prec i+n$ and $i+n \prec i+2n$). For the size of the instances we aim to solve ($20 \leq n \leq 100$), this involves an unacceptable number of maximum flow calculations (each separation routine may be called hundreds of times during the branch-and-cut algorithm). We have therefore opted not to incorporate the separation of this class of inequalities in our algorithm.

4.4.2 Branching

Three different branching rules were implemented. When the current solution is still fractional after the separation process, two child nodes are generated and added to the pool of branch-and-cut tree nodes. The *best-bound first* strategy is used to explore the pool.

Branching on a variable

Given the current fractional solution x^*, we select the variable closest to 0.5 and generate two child nodes by setting the value of that variable to either 0 or 1. In case of ties we choose the variable for which the associated arc is the longest. This is the most common rule applied in similar contexts. It is easy to implement and usually yields good results even if the branch-and-cut tree is not balanced. Indeed, setting a variable to 0 has much less consequence than setting it to 1 because the number of variables having value 1 in a feasible solution is much smaller than the number of variables having value 0. This standard branching is noted *rule A*.

We have implemented a second method for branching on variables, called *rule B*. This branching rule is slightly more flexible than rule A as it considers a wider range of variables. It consists of the following three steps:

1. store all fractional variables in a set C and determine the value that is the closest to 0.5 (noted μ),
2. remove from C all variables whose value is below $(1-\epsilon)\,\mu$ or above $(1+\epsilon)\,\mu$ (in our implementation we chose $\epsilon = 0.2$),
3. select among the remaining candidates the variable with the longest corresponding arc.

Branching on a constraint

We have also implemented a procedure for branching on a constraint. We have applied it to the SEC, but the same idea could work on other constraints such as comb inequalities. Every feasible route traverses any subset of vertices an even number of times. By traversing we mean crossing the border in the two possible directions. When we consider directed arcs, the vehicle exits and enters any proper subset that contains at least one non-transshipment vertex at most once or at least twice. If we can identify a subset $U \subset \mathcal{V}$ for which the value of the cutset $\delta^+(U)$ is close to 1.5 (where $\delta^+(U)$ represents the set of arcs with the tail in U and the head not in U), we can generate two children by enforcing $x(\delta^+(U)) \leq 1$ for the left child and $x(\delta^+(U)) \geq 2$ for the right child. In general this approach creates a branch-and-cut tree that is more balanced than when branching is performed on variables. Determining subsets on which to branch is done heuristically. We have implemented a routine that attempts to detect a subset for which the absolute difference from 1.5 is less than 0.1 (i.e., $1.4 \leq x(\delta^+(U)) \leq 1.6$). We call this branching method *rule C*. When the routine fails to find such a subset in the current fractional solution, then rule A is used.

4.5 Computational results

The branch-and-cut algorithm just described was coded in C++ and integrated in a branch-and-bound framework called *OOBB*, which stands for Object-Oriented Tools for Parallel Branch-and-Bound, currently under development at the CIRRELT in Montreal. Our code uses the sequential mode. As for the LP solver we used ILOG CPLEX 10.1. Tests were performed on an AMD Opteron Dual Core 285 2.6GHz running Linux.

To generate the instances, vertices were randomly distributed in the 500×500 square according to a discrete distribution. We have associated to the vertices a random supply and a random demand within $\{0, \ldots, m\}$ such that each object type was requested and supplied at least once. We tested with $m = 3, \ldots, 8$ object types and $n = 20, \ldots, 100$ vertices with an increment of 10 (for conciseness and without losing too much infor-

mation an increment of 20 is considered in our tables of results). For each pair (n, m) three different instances were generated leading to a total of 162 random instances.

Table 4.2 shows the average computation time in seconds for different values of n and m. As we can see the running time was not affected by increasing or decreasing the number of object types considered in the instance, the number of vertices seems to be the most influent parameter.

$n \setminus m$	3	4	5	6	7	8	Seconds
20	4	3	1	10	2	2	4
40	56	350	21	15	105	19	94
60	93	435	200	498	4903	447	1096
80	873	436	2932	237	1287	1925	1282
100	4609	795	2159	1699	1557	2561	2230
Seconds	1127	404	1063	492	1571	991	

TAB. 4.2 – Average computation times

Table 4.3 shows how the execution time was distributed during the process. The separation phase is clearly the most time-consuming step. The column heading "Other" includes the time needed to create the model, the branching procedure and miscellaneous routines inside the code. Reported values represent the average values over the 162 instances we have considered.

Solving LPs	Separating Cuts	Other
38.95%	59.88%	1.17%

TAB. 4.3 – Distribution of execution time

Table 4.4 shows the average number of drops used in the optimal solution, which corresponds to the number of times the vehicle has unloaded an object (different from the null object) at a vertex and reloaded it later during its tour. This number does not seem to be influenced by the number of object types in the instance and represented approximately 5% of the total number of vertices.

$n \setminus m$	3	4	5	6	7	8	Average
20	1	0	1	2	2	1	1.2
40	2	3	2	2	3	3	2.5
60	3	3	3	3	4	3	3.2
80	3	5	4	3	4	6	4.2
100	5	3	4	5	5	4	4.3
Average	2.8	2.8	2.8	3.0	3.6	3.4	

TAB. 4.4 – Number of drops in an optimal solution

Tables 4.5 and 4.6 show the results of our algorithm using branching rule B. The column headings are as follows. **SEC** is the number of subtour elimination constraints added to the relaxation; **Combs** is the number of comb inequalities added to the relaxation; $\boldsymbol{\pi}$ is the number of π-inequalities added to the relaxation; $\boldsymbol{\sigma}$ is the number of σ-inequalities added to the relaxation; **MTZ** is the number of times infeasible integer solutions were found and separated by adding some lifted MTZ constraints; **Gap** is the relative deviation between the solution value $\underline{z}$ obtained at the root node and the optimal solution value z^*, i.e., $(z^* - \underline{z})/z^*$; **Nodes** is the total number of nodes in the branch-and-cut tree; **Seconds** is the running time in seconds.

The π- and σ-inequalities were very useful during the resolution. Not incorporating them in the separation phase generally increases the number of branch-and-cut nodes by a factor 3 or more, and the execution time also increases. Branching rule B, which examines a pool of candidate variables, was the most efficient of the three tested rules. Branching on constraints (rule C) occasionally reduced the number of branch-and-cut nodes but at the expense of extra computation time. All results presented in this section were obtained with branching rule B.

On this set of instances we did not find any violated precedence between transshipment vertices, which is why the MTZ column has only zero entries. This can be explained by two facts. The first is that the algorithm checks for violated precedence among transshipment vertices only when the current solution is integer and integer so-

lutions rarely make use of transshipment vertices. Another reason is that the number of transshipment vertices is in general relatively small compared to the total number of vertices. However, every solution must satisfy these relationships in order to be feasible, so they have to be checked. On some other sets of instances we sometimes found violated precedences among transshipment vertices, but only rarely.

(n, m)	SEC	Combs	π	σ	MTZ	Gap	Nodes	Seconds
(20,3)	7	1	24	3	0	0.00227	3	4
(20,4)	8	1	4	0	0	0	2	3
(20,5)	5	0	1	0	0	0	1	1
(20,6)	7	5	5	2	0	0.0001	6	10
(20,7)	5	0	2	0	0	0	1	2
(20,8)	6	0	1	0	0	0	1	2
(40,3)	37	7	13	0	0	0.00003	4	56
(40,4)	40	49	98	17	0	0.00037	22	350
(40,5)	11	0	2	1	0	0	1	21
(40,6)	4	0	2	0	0	0	1	15
(40,7)	9	6	6	4	0	0.0001	6	105
(40,8)	11	1	0	0	0	0	1	19
(60,3)	36	23	33	0	0	0	1	93
(60,4)	103	46	51	1	0	0	1	435
(60,5)	47	2	23	1	0	0	2	200
(60,6)	38	11	17	7	0	0.00003	8	498
(60,7)	32	81	129	34	0	0.0001	53	4903
(60,8)	14	19	17	7	0	0	5	447

TAB. 4.5 – Detailed computational results on random instances ($20 \leq n \leq 60$)

The optimality gap at the root of the search tree is remarkably small. This yields a limited search tree exploration (less than eight nodes on average) to reach the optimal solution or to prove that the current feasible solution was optimal.

(n,m)	SEC	Combs	π	σ	MTZ	Gap	Nodes	Seconds
(80,3)	137	49	156	2	0	0.00007	2	873
(80,4)	89	3	13	0	0	0	1	436
(80,5)	138	36	64	7	0	0.00033	11	2932
(80,6)	22	1	49	0	0	0	2	237
(80,7)	33	36	71	8	0	0.00003	8	1287
(80,8)	90	41	62	7	0	0	4	1925
(100,3)	355	40	105	1	0	0	3	4609
(100,4)	51	5	39	2	0	0.00003	2	795
(100,5)	46	13	18	5	0	0	11	2159
(100,6)	66	37	19	3	0	0	4	1699
(100,7)	46	4	12	6	0	0	3	1557
(100,8)	74	11	19	10	0	0	5	2561

TAB. 4.6 – Detailed computational results on random instances ($80 \leq n \leq 100$)

4.6 Note on the mixed case

The model and the branch-and-cut algorithm presented in Sections 4.3 and 4.4 for the preemptive SP can easily be modified to handle the *mixed* case where the set of object types O is partitioned into a set O_d of droppable objects and a set O_n of non-droppable objects. To handle this situation, we only need to consider additional discardable triplets. In addition to the discardable triplets presented in Section 4.2.3 the following cases must now be considered.

Proposition 4.6.1. *If $k \in O_n$ and $b_i \neq k$, $i \in \mathcal{V}$, then the triplets $(j \in \mathcal{V}, i, k)$ are discardable.*

Proof. By definition any object type $k \in O_n$ is non-droppable so the vehicle cannot carry k to vertex i if the latter does not demand an object of that type. □

Proposition 4.6.2. *If $a_i \in O_n$ and $b_j \neq a_i$, $i, j \in \mathcal{V}$, then the triplets $(i, j \in \mathcal{V}, a_i)$ are*

discardable.

Proof. This is the symmetric case of Proposition 4.6.1. By definition, if $a_i \in O_n$ the vehicle is not allowed to carry this object type to a vertex that does not demand an object of that type. □

By setting to zero the variables associated with these triplets we ensure that the vehicle will never drop a non-droppable object. The model and the branch-and-cut algorithm presented in Sections 4.3 and 4.4 remain unchanged. With this approach we optimally solved instances with up to 100 vertices in running times similar to those reported in Table 4.5.

4.7 Conclusions

We have presented the first ever exact algorithm for the preemptive Swapping Problem on a general graph. This version of the SP turns out to be much harder to solve than the non-preemptive case studied in [15]. We have designed a mathematical model based on the structural properties of optimal solutions and elaborated a branch-and-cut algorithm to solve it. To handle the precedence relationships induced by the drop of an object at a vertex, we have used a splitting technique that allowed us to incorporate the π- and σ-inequalities, the lifted MTZ and the u-precedence constraints. Computational results show that the value of the relaxation at the root of the search tree is very close to the optimal solution value. This enables us to solve reasonably large instances within acceptable computation times.

Chapter 5

Heuristics for the Mixed Swapping Problem

Résumé. Ce chapitre propose des heuristiques pour résoudre de façon approximative la version mixte du *Problème de Repositionnement.* Dans cette version les objets sont arbitrairement *déposables* ou *non déposables.* L'algorithme sur lequel les heuristiques sont basées est constitué d'une phase constructive pendant laquelle une solution réalisable est construite, et d'une phase d'amélioration pendant laquelle des raccourcis ou des échanges d'arcs sont effectués afin de réduire le coût de la solution courante. Ces heuristiques ont été testées avec succès sur des instances contenant jusqu'à 10,000 sommets et huit types d'objet. L'écart moyen par rapport à la borne inférieure est typiquement de 1%.

Heuristics for the Mixed Swapping Problem

Charles BORDENAVE
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel GENDREAU
CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Gilbert LAPORTE
CIRRELT and Canada Research Chair in Distribution Management,
HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine,
Montréal H3T 2A7, Canada.

Article soumis pour publication dans *Computers & Operations Research*

April 2008

Abstract. In the *Swapping Problem*, to each vertex of a complete directed graph are associated at most two object types representing its supply and demand. It is assumed that for each object type the total supply equals the total demand. A vehicle of unit capacity, starting and ending its route at an arbitrary vertex, is available to carry the objects along the arcs of the graph. The aim is to determine a minimum cost route such that each supply and demand is satisfied. When some of the object types are allowed to be temporarily unloaded at some intermediate vertices before being carried to their final destination, the problem is called the *Mixed Swapping Problem*. In this paper we describe constructive and improvement heuristics which were successfully applied to randomly generated instances with up to 10,000 vertices, with an average optimality gap not exceeding 1%.

Keywords. Transportation problem, vehicle routing, heuristic

5.1 Introduction

Let $G = (V, A)$ be a complete directed graph, where $V = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) \mid i \in V, j \in V, i \neq j\}$ the arc set. Without loss of generality, vertex 1 is arbitrarily designated as a *depot*. To each vertex $i \in V$ is associated a pair of unit weight object types (a_i, b_i), where a_i is the type initially located at i (its *supply*), and b_i is the type required by i (its *demand*). The object types belong to a set $O \cup \{0\}$, where $O = \{1, \ldots, m\}$ is the set of real object types, and 0 is an additional *null* object type allowing the vertices to have only a demand or only a supply (or none). A cost matrix (c_{ij}) satisfying the triangular inequality is defined on A. A unit capacity vehicle, starting and ending its route at the depot, is available to carry the objects between the vertices of V. A route segment along which the vehicle carries the null object type (i.e., it is not loaded) is called a *deadheading*. The set O is partitioned into two subsets O_n and O_d, where O_n represents the set of *non-droppable* object types, i.e., objects that must be shipped directly from their origin to their destination, and O_d denotes the set of *droppable* object types, i.e., objects that are allowed to be temporarily dropped at some intermediate vertices on the way to their final destination. The *Mixed Swapping Problem* (MSP) consists of determining a minimum cost route allowing the vehicle to reposition the objects in such a way that all demands are satisfied.

Figures 5.1 and 5.2 represent optimal MSP solutions for two different instances defined on the unit square. The object types belonging to O_n are printed in boldface and the object type carried along an arc is shown on the arc. The depot is represented as a square. Note that the first solution of cost 6 does not use any drop because object types 1 and 2 are non-droppable, whereas in the solution of the second instance, the object of type 1 is dropped at the bottom left vertex before being carried to the upper left vertex, yielding a cost of 5.4.

The MSP was introduced by Anily and Hassin [4], who defined its main terminology and identified interesting structural properties of optimal solutions. These authors showed the problem is NP-hard by reduction to the *Traveling Salesman Problem* (TSP) and designed a 2.5-approximation algorithm for it. Anily et al. [2] have studied the

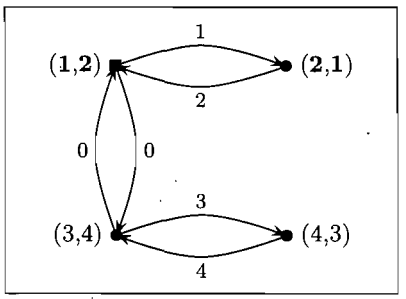


FIG. 5.1 – Optimal solution without drop

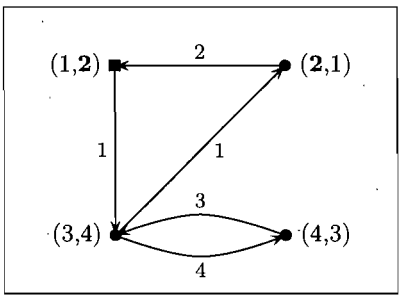


FIG. 5.2 – Optimal solution with drop

MSP on a line and proved that this particular case can be solved in polynomial time. More recently, Anily et al. [3] have shown that the SP defined on a tree is NP-hard and have provided a 1.5-approximation algorithm for this structure. They have also shown that the case where $m = 2$ can be solved in polynomial time. Bordenave et al. [15, 16] have proposed branch-and-cut algorithms for the non-preemptive and preemptive SP on a general graph. These authors were able to optimally solve instances with up to 200 vertices for the non-preemptive version, and 100 vertices for the preemptive version.

Many known routing problems are special cases of the MSP, like the *Stacker Crane Problem* (SCP) or the *Bipartite Traveling Salesman Problem* (BTSP). In the SCP, a set of arcs to be traversed by the solution is given, and the aim is to determine a minimum cost tour including these arcs. The SCP has been extensively studied. Frederickson et al. [30] have shown that the SCP on a complete graph is NP-hard and have proposed a 1.8-approximation heuristic for it. Atallah and Kosaraju [10] have considered two particular cases of the SCP where vertices are distributed along a line or along a circular shape. They have shown that these problems can be solved in polynomial time. Frederickson and Guan [28] have shown that the preemptive SCP on a tree is polynomial, but the non-preemptive SCP on a tree is NP-hard [29]. They have proposed two algorithms having worst-case performance ratios of 1.5 and 1.25. The SCP is a swapping problem (in general non-preemptive) where there exists exactly one object for each type, which means that the destination of each object is known a priori.

In the BTSP, n is even, half the vertices are black and half are white. The aim is to determine a minimum cost Hamiltonian cycle that does not visit two vertices of the

same color in succession. This problem is NP-hard. Chalasani and Motwani [17] have proposed a 2-approximation algorithm for it, based on the intersection of two specific matroids. One can easily show that this problem corresponds to a swapping problem with two object types (for this particular case, the preemptive and non-preemptive swapping problem yield the same optimal solution, which is also an optimal BTSP solution).

Our purpose is to develop heuristics for the MSP, consisting of a constructive phase followed by an improvement phase. The remainder of the paper is organized as follows. The constructive phase of the heuristics is covered in Section 5.2, while the improvement phase is presented in Section 5.3. Implementation details are discussed in Section 5.4. Computational results are reported in Section 5.5, followed by conclusions in Section 5.6.

5.2 Constructive heuristic

The purpose of this section is to describe an algorithm to construct a feasible MSP solution. It is our implementation of Algorithm 3.7 described in [4], except for Step 3.7.4 which cannot be applied if an Eulerian circuit is not available. This step has been replaced with Step 4 of our heuristic. Any feasible MSP solution is characterized by a subset of arcs, the object type carried along each arc, and the order of arc visits. If there is no drop in the solution, then the order of arc visits can be obtained by determining an Eulerian circuit, by means of the end-pairing algorithm [37], for example. The constructive heuristic described by Algorithm 1 consists of four main steps: assignment, patching, matching, and construction of an Eulerian circuit.

Definition 5.2.1. *A vertex i with $a_i = b_i$ is called a transshipment vertex.*

Since for every transshipment vertex the demand is already satisfied by its supply, our heuristic ignores all transshipment vertices, except possibly the depot, which is necessarily visited in any feasible solution even if it is a transshipment vertex.

Definition 5.2.2. *The set of non-transshipment vertices (in addition to one possible transshipment vertex i if $i = 1$) supplying an object of type k is denoted by A_k, i.e., $A_k = \{i \in V \mid a_i = k \text{ and } b_i \neq k\} \cup \{1 \mid a_1 = b_1 = k\}$. The set of non-transshipment vertices (in addition to one possible transshipment vertex i if $i = 1$) demanding an object of type k is denoted by B_k, i.e., $B_k = \{i \in V \mid b_i = k \text{ and } a_i \neq k\} \cup \{1 \mid a_1 = b_1 = k\}$.*

5.2.1 Assignment solution

By definition the demand and the supply of each vertex must be satisfied. If a solution with no drop is considered, then it consists of a set of *service paths*, i.e., a set of arcs along which the vehicle is repositioning an object from a vertex $i \in A_{a_i}$ to a vertex $j \in B_{b_j}$, where $b_j = a_i$. Therefore the service paths define a set of assignment arcs. This yields the first step of Algorithm 1, which consists of determining for each object type $k \in O \cup \{0\}$ a minimum assignment in a complete bipartite graph with vertex bipartition $\{A_k, B_k\}$. The assignment problem solution (connecting each supply to a demand), consists of a set of p simple circuits constituting connected components (Figure 5.3). If there is only one simple circuit, then it constitutes a feasible and optimal solution (see Proposition 5.2.1). Otherwise, additional arcs must be added to the current set of arcs in order to construct a feasible solution.

Proposition 5.2.1. *The assignment solution value provides a lower bound on the optimal MSP solution value.*

Proof. Let z^* be the optimal solution value. Let $\{C_t\}_{t \geq 1}$ be the collection of simple circuits obtained by solving the $m + 1$ minimum assignment problems (Step 1c), and denote by $c(U)$ the sum of the arc costs of $U \subseteq A$. In any feasible solution, an object of type k initially at vertex i (with $a_i \neq b_i$ or $i = 1$) is carried to its final destination j ($b_j = k$) either via a single arc (i, j) or via a sequence of drops at intermediate vertices between i and j. In both cases, from the triangular inequality, the total cost of this route segment is greater than or equal to c_{ij}. Therefore, the sum of the cost of each assignment circuit represents a lower bound for the problem, i.e. $z^* \geq \sum_{t \geq 1} c(C_t)$. □

Algorithm 1. Constructive heuristic

Input: $G = (V, A)$ and (a_i, b_i), $\forall i \in V$.

Output: A feasible MSP solution S and the order of arc visits.

Step 1 **Assignment**

a) Determine a minimum assignment problem in a complete bipartite graph with vertex bipartition $\{A_k, B_k\}$, $\forall k \in O \cup \{0\}$.

b) Superpose all assignment arcs to create the graph $G^0 = (V^0, A^0)$, where $V^0 = V$.

c) Identify the connected components $\{C_t\}_{t \geq 1}$ of G^0.

d) If $t = 1$, let S be the simple circuit formed by A^0. The order of arc visits is trivial since S is a simple circuit. Output S and stop.

Step 2 **Patching**

a) Select a vertex in each C_t and create the undirected graph $G^1 = (V^1, E^1)$, where each entry c^1_{ij} of the cost matrix defined on E^1 represents the minimum cost between C_i and C_j (the components containing i and j, respectively).

b) Determine a minimum spanning tree in G^1. Let $G^2 = (V^2, E^2)$ be the resulting tree, where $V^2 = V^1$.

Step 3 **Matching**

a) Identify the odd degree vertices V^3 in G^2, and create the complete undirected graph $G^3 = (V^3, E^3)$.

b) Determine a minimum perfect matching in G^3. Let $G^4 = (V^4, E^4)$ be the resulting graph, where $V^4 = V^3$ and E^4 is the set of matching edges.

Step 4 **Construction of an Eulerian circuit**

a) Direct the edges of $E^2 \cup E^4$ in such a way that $\delta^+(i) = \delta^-(i)$, $\forall i \in V^2 \cup V^4$. Let $G^5 = (V^5, A^5)$ be the resulting graph.

b) Assign object type 0 to each arc of G^5.

c) Combine A^0 and A^5 to create the graph $G^6 = (V^6, A^6)$, where $V^6 = V$.

d) Determine an Eulerian circuit in G^6 to obtain the order of arc visits, and output $S = A^6$.

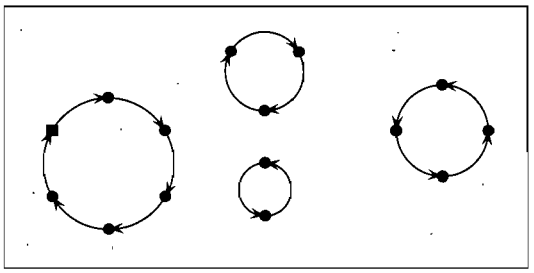


FIG. 5.3 – Assignment solution

5.2.2 Patching solution

After the p connected components in the current solution have been identified, an undirected complete graph whose vertices represent the p components is constructed. The cost of an edge linking two components C_i and C_j can be defined in several ways, for example the minimum arc cost between C_i and C_j. A minimum weight spanning tree is then determined in this graph. The arcs of the current solution and the edges of the spanning tree yield a mixed graph (Figure 5.4).

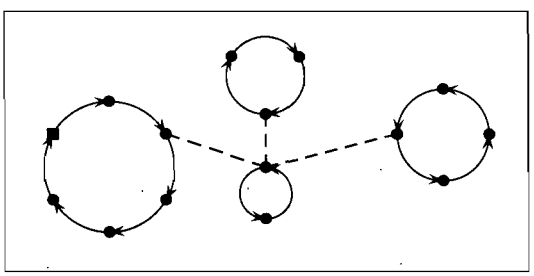


FIG. 5.4 – Patching circuits

5.2.3 Matching solution

Since any tree has at least two leaves, there exist at least two odd degree vertices in the current solution. An undirected complete graph is created on the set of odd degree vertices. It is well known that any graph has an even number of odd degree vertices, and therefore a minimum weight perfect matching can be computed on this

graph (Figure 5.5).

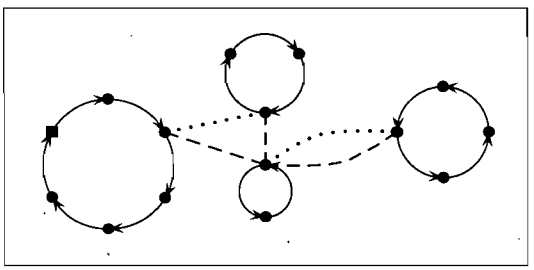


FIG. 5.5 – Matching odd degree vertices

5.2.4 Construction of an Eulerian circuit

Any feasible MSP solution is described by a set of directed arcs where each vertex has the same flow entering it and exiting it. Therefore the patching and matching edges must be directed so that for each vertex the in-degree equals the out-degree. To this end, an Eulerian cycle starting at one of the vertices incident to the patching and matching edges (arbitrarily selected) is determined. The order of edge visits in the Eulerian cycle indicates how the edges can be directed so that the degrees are balanced (Figure 5.6). Then, the null object is assigned to each of the newly created arcs, which is feasible because the vehicle can always carry the null object from any vertex to any other one.

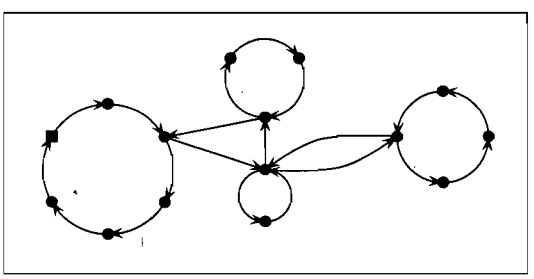


FIG. 5.6 – Directing edges

The final step consists of determining the order in which the vehicle must visit the vertices. Indeed, as shown in Figure 5.7, in which the bold arcs indicate the

vehicle circuit, a poor choice of an outgoing arc at some vertices may lead to a partial solution because some non-transshipment vertices will not be visited by the vehicle (the remaining circuit on the right-hand side is isolated). This problem is not mentioned in [4]. If one tries to construct a route by following the order of visits suggested by Theorem 3.8 of [4] (starting at the depot, traveling along a matching or patching arc, and visiting the vertices of the simple assignment circuit connected by this arc until the depot is reached), one may end up with a partial solution (see Figure 5.7). In addition, in Step 3.7.4 of [4], because the order of arc visits is not specified, it may not be possible to implement the improvement steps suggested in that paper. Indeed, since these improvements rely on consecutive arcs, they presuppose the knowledge of the order of arc visits. To avoid this, an Eulerian circuit is determined in the current solution. The sequence of arcs in the circuit indicates the order of arc visits that must be followed by the vehicle in order to visit all non-transshipment vertices (or one possible transshipment vertex if it is the depot). Note that if the cost matrix is asymmetric, then the arcs linking the connected components in the Eulerian circuit may have a cost that is different from that used to compute the minimum spanning tree in Step 2b.

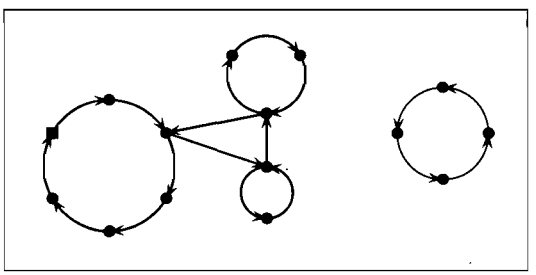


FIG. 5.7 – Partial solution with an isolated circuit

Proposition 5.2.2. *Algorithm 1 produces a feasible MSP solution with a worst-case ratio of 2.5, and this bound is tight.*

Proof. The proof given in [4] applies because the validity of this proposition depends on Steps 1, 2 and 3 only. □

Algorithm 2. Improvement heuristics

Input: A feasible MSP solution S and the order of arc visits.

Output: A feasible MSP solution S' of lower or equal cost and the order of arc visits.

Step 1 **Shortcutting**

a) Replace each pair of consecutive arcs carrying the same object type by a single arc until no such a shortcut can be made, while updating the order of arc visits. Let S' be the new solution.

Step 2 **Exchanging arcs**

a) Perform r-opt or r-r'-opt ($r' \neq r$) in S', while updating the order of arc visits.

Step 3 **Using drops**

a) Identify the set of simple circuits of deadheadings $\mathcal{C} = \{C_t\}_{t \geq 1}$ in S'.

b) If $|\mathcal{C}| \geq 1$, let (u, v) be the arc carrying object type k, that the vehicle uses to reach C_t (for a given t), and let (v, w) be the first arc of C_t. If $k \in O_d$, replace (u, v) and (v, w) with the arc (u, w) carrying k. Assign k to every other arcs of C_t.

c) Perform Step 1 in S'.

d) Output S' and the order of arc visits.

5.3 Improvement heuristics

The current solution represents a feasible MSP solution. This section described several ways to shorten the solution. These are summarized in Algorithm 2.

Proposition 5.3.1. *There exists a feasible solution that does not contain two consecutive arcs associated with the same object type.*

Proof. Follows from the triangular inequality of the cost matrix. □

5.3.1 Shortcutting

At this stage of the heuristic, since we have assigned the null object to the patching and matching arcs, shortcutting two consecutive arcs carrying the same type can be possible only with the null object and among incident arcs to vertices that belong to at least two simple circuits in the current solution (Figure 5.8). The process of shortcutting two arcs must be repeated until no more shortcut of this type can be found, since it may create two new consecutive arcs carrying the same object type.

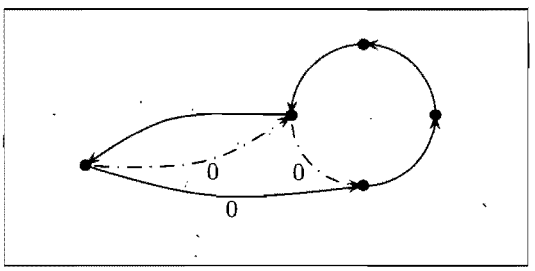


FIG. 5.8 – Shortcutting two consecutive arcs carrying the same object type

5.3.2 Exchanging arcs

A local search method can be used to improve the quality of the solution at the expense of extra computation time. Since the current solution does not contain any drop, an r-opt technique can be applied without worrying about precedence relationships. Three arc exchange techniques have been considered: 3-opt, 4-opt and 3-4-opt, which consists of repeatedly applying 3-opt and 4-opt until no improvement is possible. Similar to what happens for the directed TSP, in the MSP there is only one way, in our 3-opt and 4-opt heuristics, of reconstructing a feasible circuit when three or four arcs have been removed. This can readily be checked by enumeration. For each object type $k \in O \cup \{0\}$, r arcs carrying k are selected, and then interchanged to test whether this improves the quality of the current solution. For example, in 3-opt, three arcs $a_1 = (i_1, j_1)$, $a_2 = (i_2, j_2)$ and $a_3 = (i_3, j_3)$ carrying k are selected. If $c_{i_1 j_2} + c_{i_3 j_1} + c_{i_2 j_3} < c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3}$, then a_1, a_2 and a_3 are replaced with $a'_1 = (i_1, j_2)$, $a'_2 = (i_3, j_1)$

and $a'_3 = (i_2, j_3)$, yielding a shorter feasible solution. Similarly, in 4-opt, four arcs $a_1 = (i_1, j_1)$, $a_2 = (i_2, j_2)$, $a_3 = (i_3, j_3)$ and $a_4 = (i_4, j_4)$ carrying k are selected. If $c_{i_1 j_3} + c_{i_4 j_2} + c_{i_3 j_1} + c_{i_2 j_4} < c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3} + c_{i_4 j_4}$, then a_1, a_2, a_3 and a_4 are replaced with $a'_1 = (i_1, j_3)$, $a'_2 = (i_4, j_2)$, $a'_3 = (i_3, j_1)$ and $a'_4 = (i_2, j_4)$, yielding a shorter feasible solution. This process is repeated iteratively until no further improvement can be identified. Figures 5.9 and 5.10 illustrate a 3-opt exchange and a 4-opt exchange, respectively.

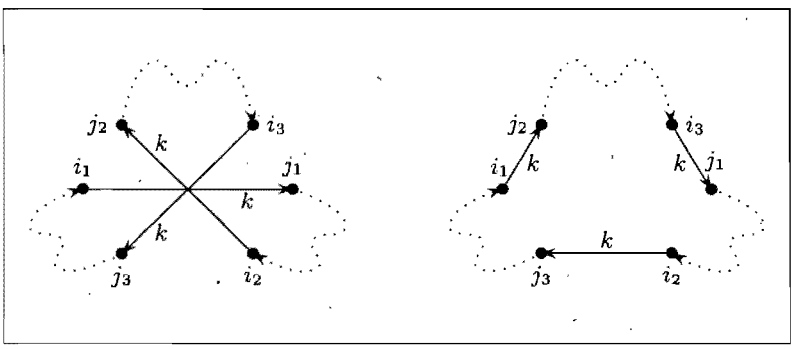


FIG. 5.9 – A 3-opt exchange

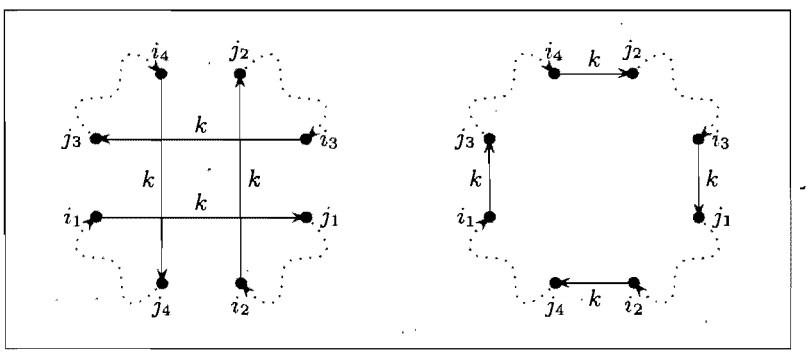


FIG. 5.10 – A 4-opt exchange

5.3.3 Using drops

So far the solution does not contain drops. Suppose there exists a simple circuit C of deadheadings in the current solution. The vehicle reaches C by an arc (u, v) carrying an object of type k, and then travels along the first arc (v, w) of C. If $k \in O_d$, then arcs

(u, v) and (v, w) can be replaced with a new arc (u, w) carrying k. Assigning k to every other arcs in C clearly leads to a new feasible solution no worse than the previous one (Figure 5.11). After doing this optimization, some consecutive arcs can possibly carry the same object type. Therefore the solution is scanned again to determine whether it can be further shortened by shortcutting two consecutive arcs carrying the same object type, as described in Section 5.3.1.

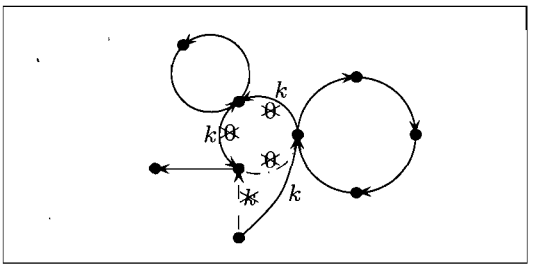


FIG. 5.11 – Shortcutting using drop ($k \in O_d$)

5.4 Implementation

The heuristics just described were implemented in C++. In this section, we discuss the various procedures contained in the heuristic, and their respective time complexity. In what follows, we call *basic* the version of the heuristic that does not incorporate the r-opt and r-r'-opt heuristics, and *full* the version that applies them.

The minimum assignment problems are solved by using the Kuhn-Munkres algorithm ([39], [44]), sometimes referred to as the Hungarian algorithm, which runs in $O(n^3)$ time. This leads to an $O(n^3 m)$ time complexity for the first step of the heuristic. This complexity order dominates all other steps of the basic heuristic. For comparison purposes we also conducted experiments in which CPLEX was used to solve the assignment problems instead of the Kuhn-Munkres algorithm. The advantage of using CPLEX is that it can solve all assignment problems simultaneously. However, memory requirement and running times are higher with CPLEX. For many operations on graphs we used Boost ([1]), which is a publicly available C++ library focussed on

data structures and graph algorithms. This package provides many routines that are both fast and easy to use. Retrieving the connected components created by the first step of the heuristic takes $O(n + n\alpha(n))$ time, where α is the inverse of Ackermann's function. Computing the minimum spanning tree over these $p \leq n/2$ components using the Kruskal algorithm ([38]) requires $O(|E| \log |E|)$ time, where E is the edge set of the complete graph defined on the p components. To solve the minimum weight perfect matching we used the *Blossom IV* code developed by Cook and Rohe [20] for Concorde ([5]). This code implements an optimized version of Edmonds algorithm ([24]) running in $O(|\widetilde{V}|^4)$ time, where $\widetilde{V}$ is the set of odd degree vertices ($|\widetilde{V}| \leq p-1$). For the computation of Eulerian circuits, we used the Hierholzer algorithm ([37]) which runs in linear time. Shortcutting arcs or using drops can also be performed in linear time. The r-opt and r-r'-opt heuristics are pseudo-polynomial. Since this process is time consuming (each step of r-opt requires $O(n^r)$ time and the number of steps can be high), we have applied these improvement steps only on instances containing fewer than 1000 vertices.

5.5 Computational results

Three sets of instances were generated as follows. Each set contains random geometric instances in which n vertices are located in a 500×500 square according to a uniform discrete distribution. Each vertex is associated with a random supply and demand such that for each object type the total supply equals the total demand. The number $|O_d|$ of droppable object types was randomly selected between 0 and $|O|$. We have tested the heuristic on values of n ranging from 100 to 10,000 and on values of m ranging from 3 to 8. The reported results correspond to the average over these three sets. Tests were performed on an AMD Opteron Dual Core 285 2.6GHz (1 GB RAM was required for solving the large instances).

Tables 5.1 and 5.2 report computation times (in seconds) and optimality gaps with respect to the assignment lower bound (in percentage) for different values of n and m, for the basic heuristic. Instances with few object types are more difficult to solve. This can be explained by the fact that the assignment problems, which represent the

$n \setminus m$	3	4	5	6	7	8	Average
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	0.2	0.1	0.1	0.1	0.1	0.0	0.1
500	3.8	2.2	1.5	1.1	0.8	0.6	1.7
1000	48.0	23.5	16.0	11.0	8.4	5.8	18.8
2000	434.0	290.4	175.6	130.4	81.7	67.2	196.5
5000	14041.6	7990.2	4940.9	3302.2	2238.6	1534.6	5674.7
10,000	161343.2	101560.0	73437.0	44575.1	31084.9	20461.3	72076.9
Average	25124.4	15695.2	11224.4	6860.0	4773.5	3152.8	

TAB. 5.1 – Computation times for the basic heuristic (in seconds)

most time consuming part of the basic heuristic, are harder to solve. Indeed, for a fixed number of vertices, decreasing the number of object types increases the number of possible assignments for a given object.

As shown in Table 5.2 the optimality gap is remarkably small. It represents on average only 0.9% with respect to the lower bound provided by the assignment solution value. It tends to become smaller as the instance size become larger. The number of object types also influences the size of the optimality gap. If we consider small values of m, the assignment solution is typically formed by a large collection of simple circuits, each containing very few vertices. Since a solution generated by the heuristic is based on the assignment solution, the application of the patching and matching procedures produces solutions for which the optimality gap is larger.

Tables 5.3 and 5.4 provide a comparison of the different optimization methods in terms of average computation time and average optimality gap. Since these methods are time consuming we only tested them for relatively small values of n. We can see that 3-opt improves the optimality gap by about 25% with only a relatively small increase in computation time. This algorithm seems to be a good choice for small and medium size instances. The 4-opt heuristic is far less attractive since it generates larger optimality gaps and has very high computation times. The method yielding the

$n \setminus m$	3	4	5	6	7	8	Average
100	2.922	2.338	0.937	1.690	0.671	0.633	1.532
200	2.286	1.923	1.047	1.149	0.470	0.721	1.266
500	2.116	1.317	0.425	0.613	0.638	0.414	0.921
1000	1.855	1.275	0.576	0.689	0.650	0.331	0.896
2000	1.712	0.684	0.820	0.397	0.389	0.267	0.712
5000	1.859	0.767	0.546	0.459	0.381	0.297	0.718
10,000	1.644	0.716	0.632	0.480	0.240	0.200	0.652
Average	2.056	1.289	0.712	0.782	0.491	0.409	

TAB. 5.2 – Optimality gaps for the basic heuristic with respect to the assignment lower bound (in percentage)

n	Basic	3-opt	4-opt	3-4-opt
100	0.0	0.0	0.1	0.2
200	0.1	0.3	3.3	2.2
500	1.7	7.4	246.7	128.3
1000	18.8	95.7	4596.1	2736.2
Average	5.1	25.8	1211.5	716.7

TAB. 5.3 – Comparison of computation times for the full heuristic (in seconds)

smallest optimality gaps was the 3-4-opt heuristic, which combines 3-opt and 4-opt, but with this method running times also increase quickly with n. Performing 4-opt after 3-opt sometimes allows to escape from the current local minimum. On average, the 3-4-opt improvement heuristic yields a gap reduction of about 32% over the basic algorithm. Since 3-opt often improves the solution, there are fewer opportunities for 4-opt to do so, and then the overall process takes in general less time than applying only 4-opt. It should be noted that for each of these optimization techniques, the number of object types drastically affects the computation time. Instances for which m is small take more time since the number of triplets or quadruplets to consider is higher. For

n	Basic	3-opt	4-opt	3-4-opt
100	1.532	1.251	1.346	1.147
200	1.266	0.922	0.979	0.829
500	0.921	0.643	0.678	0.592
1000	0.896	0.639	0.680	0.585
Average	1.154	0.864	0.921	0.788

TAB. 5.4 – Comparison of optimality gaps for the full heuristic with respect to the assignment lower bound (in percentage)

example, we noticed that applying 3-opt on a 1000-vertex instance with three object types increases by a factor 10 or more the time needed to solve an instance of the same size but containing eight object types. Finally tests have shown that the proportion of droppable object types has no significant effect on the performance of our heuristics.

5.6 Conclusions

We have provided a complete description of a constructive and several improvement heuristics for the *Mixed Swapping Problem*. These heuristics were successfully applied to large instances containing up to 10,000 vertices. The average optimality gap is remarkably small, typically less than 1%.

Chapitre 6

Conclusion

Cette thèse a présenté plusieurs algorithmes de coupes et branchements pour résoudre optimalement le *Problème de Repositionnement*. La version préemptive du PR s'est révélée être la plus difficile, mais l'agorithhme développé pour la résoudre a permis, moyennant de très légères modifications, de résoudre également la version mixte. La taille des problèmes résolus jusqu'à optimalité est encourageante compte tenu de la complexité du PR. Il n'existait pas jusqu'à présent de méthodes exactes pour résoudre le PR dans le cas général. Le travail présenté ici représente donc la première étude proposant de telles méthodes pour le PR. Les heuristiques proposées permettent de résoudre de larges instances et les solutions générées sont en moyenne très proches des solutions optimales.

Les algorithmes exacts présentés dans cette thèse pourraient être améliorés en intégrant d'autres types d'inégalités valides lors de la phase de séparation. On peut citer par exemple les inégalités d'étoiles, les inégalités multi-étoiles, les inégalités de cliques, les inégalités de chemins, pour la version préemptive, et y ajouter les contraintes d'ordre généralisé et d'autres inégalités valides développées pour le *Problème du Voyageur de Commerce avec Contraintes de Préséance*, pour la version préemptive. Il faut toutefois noter qu'incorporer un grand nombre de classes d'inégalité dans un algorithme de coupes et branchements peut ralentir de façon importante le processus car la séparation

de ces inégalités prend un temps considérable, alors que le branchement est parfois plus bénéfique.

De nombreuses extensions ou variantes du PR peuvent être envisagées. On peut par exemple considérer une version du PR dans laquelle on disposerait de k véhicules pour effectuer l'ensemble des repositionnements. Des tests préliminaires encourageants ont montré que les algorithmes présentés dans cette thèse peuvent être adaptés sans trop de difficulté pour résoudre cette variante du problème. On peut également envisager le cas d'un véhicule de capacité Q qui lui permettrait de transporter plusieurs objets à la fois. Finalement on peut imaginer une version du PR dans laquelle les sommets offrirait et demanderait plus d'une unité de chaque objet, voire éventuellement plusieurs objets de type différent.

Bibliographie

[1] Boost C++ libraries, 2001–. `http://www.boost.org`.

[2] S. Anily, M. Gendreau, and G. Laporte. The swapping problem on a line. *SIAM Journal on Computing*, 29 :327–335, 1999.

[3] S. Anily, M. Gendreau, and G. Laporte. The preemptive swapping problem on a tree. Submitted for publication, 2006.

[4] S. Anily and R. Hassin. The swapping problem. *Networks*, 22 :419–433, 1992.

[5] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. Concorde TSP solver, 1995–. `http://www.tsp.gatech.edu/concorde.html`.

[6] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems.* Ph.D. thesis, Technische Universität Berlin, Germany, 1995.

[7] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. On identifying in polynomial time violated subtour elimination and precedence forcing constraints for the sequential ordering problem. In *Proceedings of the First Integer Programming and Combinatorial Optimization Conference*, pages 19–28. Waterloo, Canada, 1990.

[8] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3 :25–42, 1993.

[9] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asym-

metric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17 :61–84, 2000.

[10] M.J. Atallah and S.R. Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17 :849–869, 1988.

[11] E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68 :241–265, 1995.

[12] M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36 :192–201, 1988.

[13] A. Baltz and A. Srivastav. Approximation algorithms for the Euclidean bipartite TSP. *Operations Research Letters*, 33 :403–410, 2004.

[14] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems : A classification scheme and survey. *TOP*, 15 :1–31, 2007.

[15] C. Bordenave, M. Gendreau, and G. Laporte. A branch-and-cut algorithm for the non-preemptive swapping problem. Submitted for publication, 2008.

[16] C. Bordenave, M. Gendreau, and G. Laporte. A branch-and-cut algorithm for the preemptive swapping problem. Submitted for publication, 2008.

[17] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28 :2133–2149, 1999.

[18] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.

[19] V. Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical Programming*, 5 :29–40, 1973.

[20] W.J. Cook and A. Rohe. Computing minimum weight perfect matchings. *INFORMS Journal on Computing*, 11 :138–148, 1997.

[21] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54 :573–586, 2006.

[22] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33 :1–27, 1985.

[23] M. Desrochers and G. Laporte. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10 :27–36, 1991.

[24] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17 : 449–467, 1965.

[25] J. Edmonds. Submodular functions, matroids and certain polyhedra. In *Proceedings of the Calgary International Conference*, pages 69–87. Calgary, Canada, 1970.

[26] J. Edmonds. Matroid intersection. *Annals of Discrete Mathematics*, 4 :39–49, 1979.

[27] L.F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37 :236–249, 1988.

[28] G.N. Frederickson and D.J. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21 :1130–1152, 1992.

[29] G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15 :29–60, 1993.

[30] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7 :178–193, 1978.

[31] G. Ghiani, G. Laporte, and F. Semet. The black and white traveling salesman problem. *Operations Research*, 54 :366–378, 2006.

[32] A. Goldberg and R. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35 :921–940, 1988.

[33] M. Grötschel and M.W. Padberg. On the symmetric traveling salesman problem I : Inequalities. *Mathematical Programming*, 16 :265–280, 1979.

[34] J. Hao and J.B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Orlando, Florida, 1992.

[35] H. Hernández-Pérez and J.J. Salazar González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145 :126–139, 2004.

[36] H. Hernández-Pérez and J.J. Salazar González. The one-commodity pickup-and-delivery traveling salesman problem : Inequalities and algorithms. *Networks*, 50 : 258–272, 2007.

[37] C. Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechnung zu umfahren. *Mathematische Annalen*, 6 :30–32, 1873.

[38] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[39] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quaterly*, 2 :83–97, 1955.

[40] A.N. Letchford, R.W. Eglese, and J. Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94 :21–40, 2002.

[41] J. Lysgaard. CVRPSEP package, 2004–. `http://www.hha.dk/~lys/`.

[42] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100 : 423–445, 2004.

[43] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the ACM*, 7 :326–329, 1960.

[44] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5 :32–38, 1957.

[45] D. Naddef and S. Thienel. Efficient separation routines for the symmetric traveling salesman problem I : General tools and comb separation. *Mathematical Programming*, 92 :237–255, 2002.

[46] K.S. Ruland. *Polyhedral Solution to the Pickup and Delivery Problem.* Ph.D. thesis, Sever Institute of Washington University, St. Louis, Missouri, 1994.

[47] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem : Faces and branch-and-cut algorithm. *Computers and Mathematics with Applications*, 33 :1–13, 1997.