INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Département Génie Mathématique

Stage de 3GM

Optimisation du damage d'une station de ski

11 juin au 31 juillet 2019

Théo Guyard

theo.guyard@insa-rennes.fr

Organisme d'accueil : **INSA Rennes et IRMAR**

Maître de stage : **Jeremy Omer**

Enseignant référent : **Ayse Nur Arslan**

Résumé

Chaque soir dans les stations de ski, des dameuses partent pour remettre en état la station. Le but est de damer les pistes qui ont été utilisées la journée précédente. L'objectif de ce stage est de trouver une manière optimale d'effectuer le damage d'une station de ski à partir d'un plan de la station et d'un nombre de véhicules disponibles. L'approche utilisée est la modélisation sous forme de graphe de la station. Cette approche permet de lier le problème de damage aux problèmes classiques de routage. Suivant les paramètres du problème, on sera amené à résoudre un type de problème de routage en particulier. Pour les problèmes les plus simples, une modélisation par PLNE sera décrite puis implémentée pour être résolue par un solver. Dans le cas le plus complexe, un algorithme de Branch-and-Price sera proposé. L'objectif de ce stage est également de se familiariser avec le monde de la recherche, les publications scientifiques, d'apprendre à effectuer des recherches bibliographiques et de commencer à faire de la recherche sur un problème encore ouvert. Il a été effectué au sein du département Génie Mathématique de l'INSA Rennes.

Mots clés : Graphes, Programmes linéaires, PLNE, Problème du postier Chinois, CARP, Branch-and-Price

Remerciements

Je tiens tout particulièrement à remercier M. Omer et Mme Arslan de m'avoir proposé ce stage et guidé durant sa réalisation ainsi que M. Gilbert pour avoir contribué à la relecture de ce rapport.

Table des matières

1 Introduction

1.1 Problème

Pour attirer des clients, une station de ski se doit d'avoir des pistes avec une bonne neige. Pour entretenir les pistes, les stations sont damées chaque soir. Des dameuses passent sur les pistes afin de ré-étaler et renouveler la neige. Cette opération est effectuée sur une partie ou sur la totalité des pistes de la station. Le problème qui va être traité lors de ce stage consiste à trouver une manière optimale de damer les pistes d'une station de ski. Pour cela, on aura à disposition un plan des pistes et un nombre de véhicules disponibles. On connaîtra également certaines informations quand au coût des opérations de damage sur les pistes et quand aux caractéristiques des dameuses. L'objectif sera de trouver le chemin que doit effectuer chaque dameuse pour la prochaine session de damage. Il n'est pas obligatoire de toutes les utiliser. Par la suite, ce problème sera appelé GOP (Groomer Optimisation Problem).

Dans une station de ski, toutes les dameuses sont entreposées dans des hangars appelés "dépôts" desquels elles partent et auxquels elles doivent revenir en fin de service. On suppose qu'une dameuse est toujours affectée à un unique dépôt. Toutes les pistes et les chemins praticables par les dameuses sont connectés entre eux par au moins une jonction de telle manière qu'on puisse atteindre n'importe quel endroit de la station depuis tous les dépôts. Néanmoins, certaines pistes ne sont pas praticables dans les deux sens pour les dameuses.

Dans les petites stations possédant une unique dameuse, celle-ci doit être capable de damer toutes les pistes en une seule session de damage. Dans les stations possédant plusieurs dameuses, il faut assigner un chemin pour chaque dameuse afin qu'au terme d'une session de damage, toutes les pistes qui avaient besoin d'entretien aient été damées. Pour éviter qu'une dameuse soit surchargée de travail par rapport aux autres, chaque dameuse possède une charge maximale de travail qu'elle ne peut pas dépasser.

Pour définir l'optimalité du GOP, nous considéreront qu'on peut définir pour chaque piste un demande, un coût de passage sur la piste et un coût de damage de la piste. Si la demande d'une piste est positive, alors elle doit être damée. On pourra donc associer un coût à chaque chemin effectué par les dameuses en fonction des pistes qu'elles dament ou non. Le damage optimal sera celui qui minimise ces coûts pour toutes les dameuses et qui permet que toutes le pistes avec une demande positive soient damées.

1.2 Formalisation du problème

1.2.1 Modélisation

Nous allons modéliser le GOP sous forme de graphe. L'ensemble des pistes formera les arêtes du graphe et les jonctions entre les pistes formeront ses sommets. Chaque arête aura une demande ainsi qu'un coût de passage et un coût de damage associé.

On note $G = (S, A' \cup E)$ le graphe associé aux pistes où S est l'ensemble des jonctions sur les pistes et U est l'ensemble des pistes ou chemins praticables pour les dameuses. Une piste sera représentée par un arc (de A') si la piste est praticable dans un seul sens ou par une arête (de E) si elle est praticable dans les deux sens. On dira d'une piste qu'elle est "requise" si elle a une demande positive et "servie" si elle est damée par une dameuse. Il sera possible que plusieurs dameuses empruntent la même piste lors d'une session de damage. Par conséquent, une dameuse pourra passer sur une piste sans la damer. Comme les dameuses devront partir et revenir au dépôt qui leur est associé, elles devront effectuer des cycles sur G. Dans le cas avec plusieurs dameuses, on leur assignera une capacité maximale. La somme des demandes sur les pistes traitées par une dameuse ne devra pas excéder cette capacité maximale afin de ne pas surcharger la dameuse de travail.

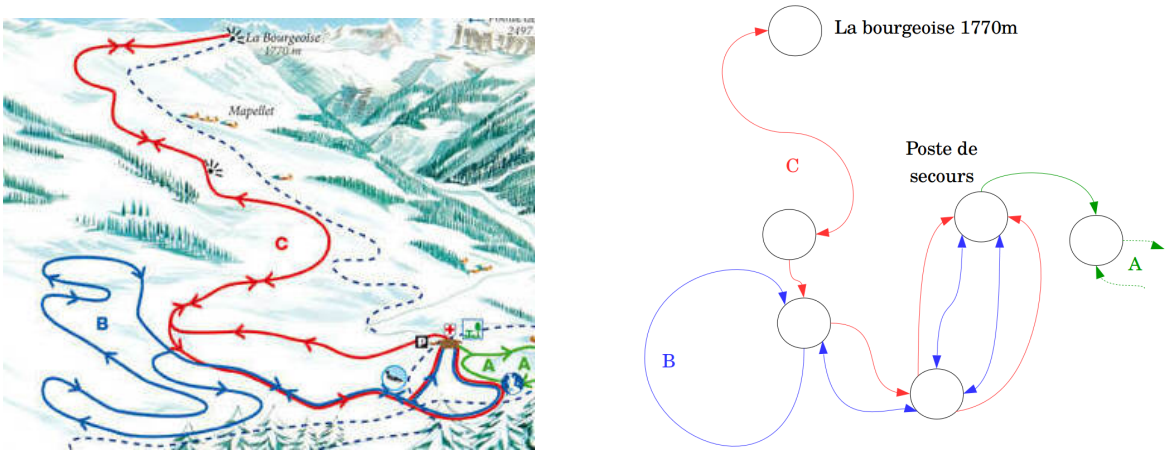


FIGURE 1 – Modélisation de pistes sous forme de graphe (ici un graphe mixte)

1.2.2 Gestion des coûts et de la demande

Dans le modèle, on suppose qu'on peut associer une demande ainsi qu'un coût de damage et un coût de passage sur chaque piste. Nous proposons ici une manière d'estimer ces valeurs pour une piste.

Chaque soir, la station fait un point sur la qualité des pistes et détermine celles qui sont à damer. La charge de travail n'est pas forcément égale pour toutes les pistes. Celle-ci peut dépendre de la quantité de neige à traiter, de la qualité de la neige, etc... Dans notre modèle, on effectuera une somme pondérée de ces différentes charges de travail qu'on regroupera sous une unique valeur : la demande de la piste. On pourra également se baser sur des caractéristiques de la piste (longueur, pente, etc...) ainsi que sur le prix du carburant ou le prix d'entretien d'une dameuse pour calculer les coûts de damage et de passage sur une piste.

Exemple :

Prenons l'exemple d'une piste très pentue sur laquelle on doit damer $50m^3$ de neige. Les équipes de la station ont calculé que la consommation de carburant est de 1 L pour $10m^3$ de neige. Cette piste mesure 2 km et le carburant coûte 5 € par litre. De plus, on ne peut que monter la piste avec une dameuse donc la consommation de carburant sera multipliée par 1.5. Enfin, la consommation pour le passage sur une piste est de 1 L par kilomètre.

Demande :

$$d = 50\ m^3$$

Coût de passage :

$$p = 2\ km \times\ 1.5 \times 1\ \frac{L}{km}\ \times 5\frac{€}{L} = 15€$$

Coût de damage :

$$c = p + \frac{1\ L}{10\ m^3} \times d \times 5\frac{€}{L} = 40€$$

1.2.3 Caractéristiques propres au problème

Deux caractéristiques propres au GOP pourront justifier les choix qui seront faits par la suite pour résoudre le problème.

Taille
Une des principales caractéristiques du problème est la taille. En effet, en restant dans le cadre des stations de ski, on sera quasiment sûr que le graphe obtenu en modélisant les pistes de dépassera pas une certaine taille. Il sera rare d'avoir plus de 100 sommets en modélisant le graphe.

Coûts
Le damage des stations de ski est assez coûteux. Rajouter une dameuse ou effectuer un chemin qui n'est pas optimal peut être très pénalisant. Donner une solution approchée du problème n'est donc pas une bonne solution en pratique. On cherchera toujours une solution exacte et, si possible, une qui minimise le nombre de dameuses utilisées.

1.3 Lien avec les problèmes de routage

Suivant les contraintes qu'on se fixe, on pourra faire un lien avec les problèmes classiques de routage.

Problème du postier Chinois (CPP) : Trouver, dans un graphe connexe non-orienté, un cycle de longueur minimale passant par toutes les arêtes.

Problème du postier Chinois non orienté (DCPP) : Trouver, dans un graphe fortement connexe orienté, un cycle de longueur minimale passant par tous les arcs.

Problème du postier Chinois mixte (MCPP) : Trouver, dans un graphe fortement connexe mixte, un cycle de longueur minimale passant par toutes les arêtes et tous les arcs dans au moins un sens.

Problème du postier Chinois rural (RCPP, DRCPP, MRCPP) : Le CPP, DCPP et MCPP s'étendent respectivement au problème du postier Chinois rural (RCPP), au problème du postier Chinois rural orienté (DRCPP) et au problème du postier Chinois rural mixte (MRCPP) dans le cas où on doit seulement passer par un sous-ensemble d'arcs/arêtes.

Capacited Arc Routing Problem (CARP) : Le problème de couverture par arcs avec capacité (CARP) consiste à placer sur les arêtes d'un graphe non orienté une certaine demande en plus de leur poids et de trouver un ensemble (de taille fixée) de cycles de longueur minimale partant d'un nœud "dépôt" qui permet de couvrir toutes les arêtes avec une demande positive. On veut également que chaque cycle ne dépasse pas une certaine capacité maximale (qui correspond à la somme des demandes des arcs sur lesquels le cycle passe). Ce problème peut s'apparenter à un problème de collecte de déchets dans une ville où on doit débarrasser tous les habitants de leurs déchets tout en respectant la quantité maximale de déchets pouvant être mise dans un camion poubelle. Lorsqu'on cherche une solution sur un graphe mixte, on note ce problème MCARP. Si on s'autorise à avoir plusieurs nœuds dépôt sur un graphe mixte, ce problème est noté MD-MCARP (Multi-Dépôt MCARP).

Dans le cas du GOP avec une unique dameuse, on suppose que la dameuse peut s'occuper de la totalité des pistes toute seule et on ne prend donc pas en compte sa capacité. Si l'ensemble des pistes est requis, on sera dans le cas du CPP, DCPP ou MCPP suivant le type des pistes. Si seulement une partie des pistes est requise, on sera dans le cas du RCPP, DRCPP ou MRCPP à l'exception près qu'il faut passer par le dépôt dans le cycle solution. Dans le cas d'un GOP avec plusieurs dameuses, on peut leur associer à chacune une capacité maximale afin qu'une dameuse ne soit pas surchargée par rapport aux autres. Le problème reviendra donc à un problème de CARP. On traitera tous les problèmes avec plus d'une dameuse comme des problèmes de MCARP (CARP pour un graphe mixte) car dans la plupart des cas, le graphe obtenu lors de la modélisation est mixte. Enfin, si la station possède plusieurs dépôts de dameuse, on sera dans le cas du MD-MCARP.

1.4 Méthode de résolution

Lors de la résolution du GOP, on cherchera dans un premier temps à trouver à quel problème de routage on peut s'apparenter. Les problèmes de postier Chinois (non ruraux) sont P-durs alors que le MCARP et le MD-MCARP sont NP-durs. Choisir le bon type de problème permettra de gagner du temps de calcul. On va ensuite résoudre un programme linéaire de manière optimale. Celui-ci permettra de trouver combien de fois chaque véhicule doit passer par chaque arc/arête et lesquels il doit servir. On pourra ensuite reconstruire le chemin de chaque dameuse à partir des arcs/arêtes sur lesquels elle passe. Le choix de modéliser ce problème sous forme de programme linéaire est motivé par le fait qu'on veut pas de solution approchée. Les résolutions de programmes linéaires peuvent prendre beaucoup de temps lorsqu'il y a un nombre important de variables. Dans notre cas, celles-ci seront liées au nombre d'arcs/arêtes dans le graphe qui ne sera pas de taille très importante. On peut donc espérer des temps de calcul convenable.

Pour les cas avec un seul dépôt, on pourra utiliser une interface de programmation mathématique qui nous permettra de décrire notre programme linéaire puis on utilisera un solver pour le résoudre. Pour les cas de problèmes du postier Chinois ruraux, on n'implémentera pas un algorithme spécifique. On utilisera la modélisation du MCARP et un seul véhicule sera donné en paramètre. Les problèmes du postier Chinois ruraux et le MCARP étant tous deux NP-durs, on ne gagnerait pas beaucoup de temps de calcul en modélisant différemment les problèmes de postier Chinois ruraux, surtout si on doit inclure la contrainte de passage par le dépôt. Le cas avec plusieurs dépôts de dameuses est plus compliqué. Dans ce cas, on proposera un algorithme de Branch-and-Price pour résoudre GOP.

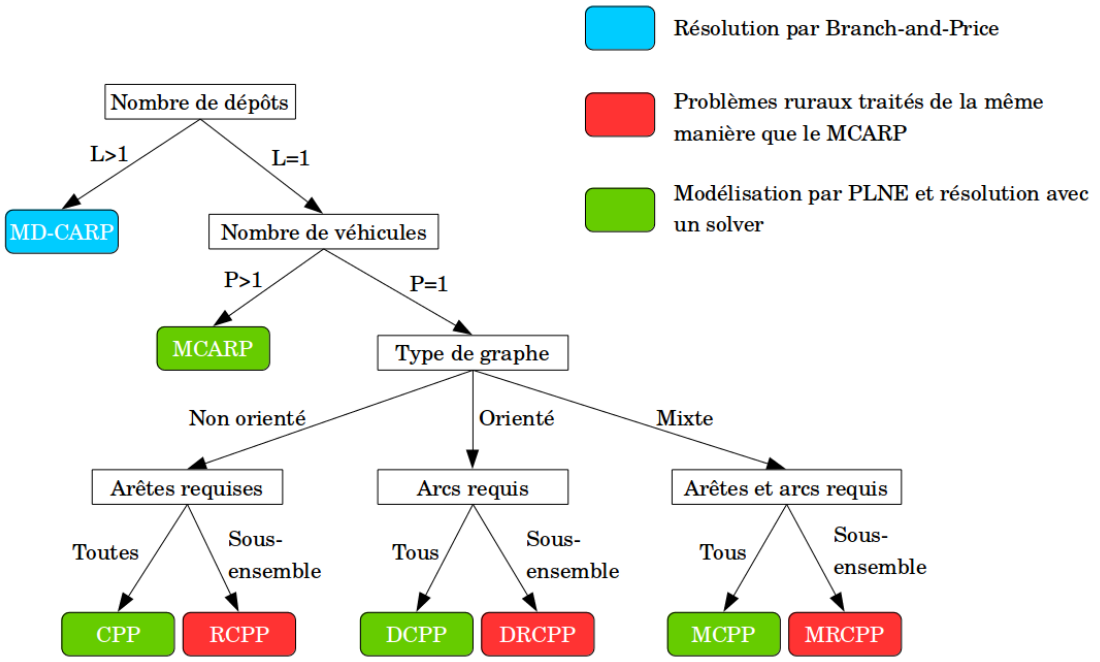


FIGURE 2 – Classification du problème de damage et méthodes de résolution

2 Recherches bibliographiques

Les problèmes de routage sont assez classiques et on retrouve cette modélisation pour de nombreux problèmes réels. Le premier problème de routage est lié au problème des sept ponts de Königsberg [8]. Ce problème consiste à trouver un chemin passant une unique fois par chacun des sept ponts de la ville de Königsberg. Euler a démontré en 1736 que ce problème n'avait pas de solution, mais que si on permettait de passer plusieurs fois par les ponts, on pouvait trouver un chemin de longueur minimale pour ce problème.

En 1962, Kwan Mei-Ko a été le premier à proposer une solution au problème plus général de couverture par les arêtes dans un graphe connexe non-orienté [25]. C'est à ce mathématicien Chinois que le CPP doit son nom. Kwan Mei-Ko montre que ce problème revient à trouver un couplage parfait de poids minimum entre les sommets de degrés impair afin de rajouter des arêtes au graphe pour le rendre Eulérien. Il faudra ensuite trouver un chemin Eulérien dans le nouveau graphe, à l'aide par exemple de l'algorithme de Hierholzer [2]. À cette époque, il n'existe pas d'algorithme polynomial pour le problème de couplage parfait de poids minimum.

J. Edmonds propose en 1965 l'algorithme du "Blossom" [6] qui permet de résoudre un problème de couplage maximal en un temps polynomial. C'est le premier algorithme polynomial pour résoudre un problème de couplage. Par la suite, des versions plus élaborées de cet algorithme sont développées pour résoudre tout type de problème de couplage. On peut notamment citer l'algorithme "Blossom V" [15] développé en 2010 par Kolmogorov qui est la dernière version permettant de résoudre efficacement le problème du couplage parfait de poids minimum. À partir des années 1970, le CPP est donc classé comme un problème P-dur.

Par la suite, de nombreuses variantes au CPP ont été étudiées. Le DCPP ou le MCPP sont des dérivés du CPP sur d'autres types de graphes. Les problèmes de type "rural" sont des CPP où on doit couvrir simplement un sous-ensemble des arêtes. Les premiers algorithmes de résolution exacte, introduits en 1973 par J. Edmonds, traitent le DCPP comme un problème de flot et il transforment le MCPP pour pouvoir le traiter de la même manière [7] [20] [10]. Les problèmes de type "rural" sont, quand à eux, de type NP-dur. Il faut attendre 1981 pour avoir les premières formulations exactes sous forme de programme linéaire de ces problèmes [4]. Celles-ci utilisent un nombre exponentiel de contraintes. Les problèmes dans lesquels on doit trouver une couverture par les arêtes d'un graphe pour un nombre fixé de chemins sont aussi de classe NP-dur. Des heuristiques sont alors développées à partir des années 2000 pour résoudre ce type de problème en un temps raisonnable. Il existe différentes approches. On peut partir d'une solution du CPP puis découper le cycle en sous-cycles ou alors utiliser des méthodes de cluster [31].

Au début des années 2000, le taille grandissante des réseaux de distribution ou des réseaux de communication font émerger des problèmes de type CARP [11]. Ces problèmes consistent à trouver sur un graphe un ensemble de chemins couvrant un sous-ensemble des arêtes du graphe tout en respectant certaines contraintes sur chaque chemin. Les réseaux pouvant très simplement être modélisés sous forme de graphe, ces problèmes sont encore aujourd'hui au cœur de nombreux sujets de recherche. Wøhlk donne une liste et un historique des problèmes et des solutions liées au CARP [36].

Le CARP est NP-dur, on peut le relier au problème du voyageur de commerce en transformant le graphe [9]. La résolution par programme linéaire en nombres entiers est quasiment systématiquement utilisée lorsqu'il faut trouver une solution exacte. Une première formulation est faite en 1981 par Golden et Wong mais nécessite un nombre exponentiel de contraintes [11]. Belenguer et Benavent ont beaucoup travaillé sur les formulations du CARP pour les simplifier. Ils proposent en 2003 un modèle avec une seule variable de décision par arc, mais avec un nombre de contraintes toujours exponentiel [3] en se basant sur les méthodes des plans coupants. Le problème a été étendu par la suite à tout type de graphes sous la dénomination de MCARP et la première formulation compacte de ce problème a été publiée en 2010 [17]. Celle-ci introduit une nouvelle variable de flot. Ceci permet d'avoir un nombre

polynomial de variables. C'est la première formulation qui donne des résultats satisfaisants sur des jeux de données de taille moyenne [35]. Cette formulation compacte est étendue au MD-CARP (plusieurs dépôts disponibles) en 2015 [16]. Des algorithmes de Branch-and-Price ont également été développés spécifiquement pour le CARP afin de résoudre ce problème sur de plus grosses instances [24].

En parallèle, des heuristiques sont développées pour les cas de graphes de grande dimension et où une solution exacte n'est pas forcément nécessaire. Ces problèmes sont souvent liés à l'industrie ou aux problèmes de réseaux de distribution/collecte dans les grandes villes. L'exemple de la collecte de déchets dans les villes est d'ailleurs l'exemple le plus courant du MCARP/MD-CARP. Ces heuristiques sont basées sur plusieurs concepts comme le clustering [14], les recherches avec tabou [1] ou les algorithmes de colonies de fourmis [18].

Une bibliographie très complète des problèmes de routage a été proposée en 2017 par M. Mourao [29]. Elle sépare les problèmes de routage en deux catégories : unique véhicule ou plusieurs véhicules. Dans le cas avec un seul véhicule, on retrouve tous les problèmes du postier Chinois ainsi que ses variantes (introductions de contraintes, changement de type de graphe ...). Dans le cas avec plusieurs véhicules, on retrouve le problème classique du CARP ainsi que des variantes où on rajoute des dépôts ("k-ARPs"), où on introduit de nouvelles valeurs sur les arcs ("With profit") et où on ne doit pas simplement trouver des tournées mais aussi des localisations optimales de dépôt, des périodes optimales ("Tactical") ... Cette bibliographie nous donne également toutes les applications que peuvent avoir ce type de problèmes.

Dans la prochaine section, nous détaillerons certaines techniques basiques de résolution de problèmes du postier Chinois. Pour notre problème, nous utiliserons des résolutions à base de programmes linéaires en nombres entiers qui seront détaillées dans la section suivante.

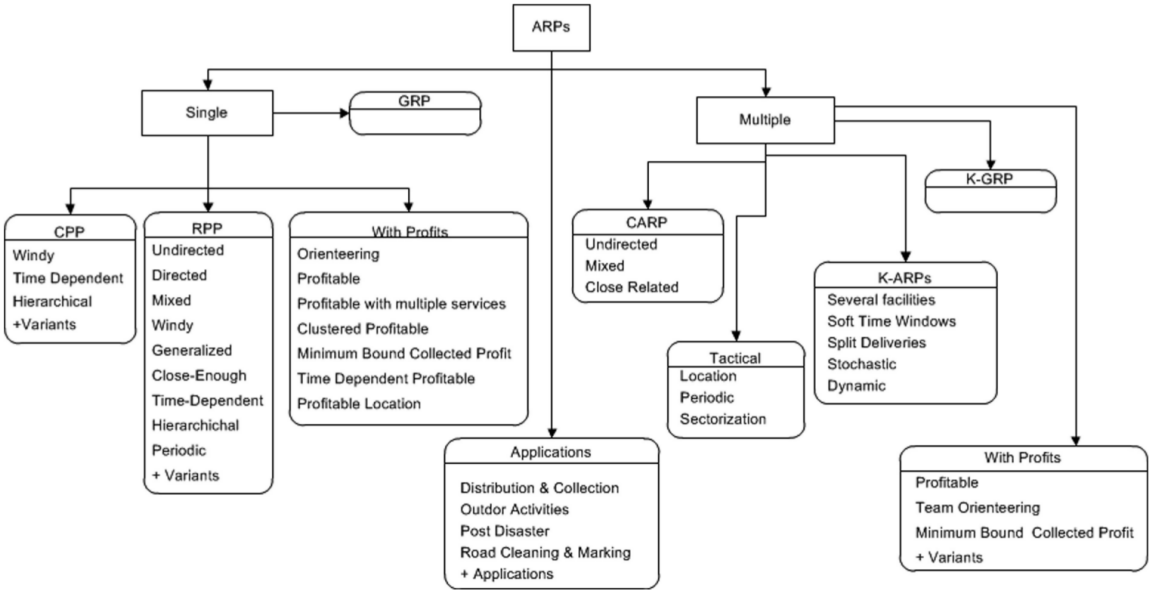


FIGURE 3 – Vue d'ensemble des problèmes de routage par M. Mourao

3 Compléments sur les problèmes de routage

Cette section rend compte des recherches bibliographiques effectuées au début du stage qui ne se sont pas avérées utiles dans la résolution du problème, mais qui ont permis une bonne compréhension des concepts qui y sont liés. Des algorithmes explicites ou heuristiques pour résoudre cas des problèmes du postier Chinois seront présentés.

3.1 Problème du postier Chinois

Soit $G = (S, U)$ un graphe non-orienté connexe, on cherche à résoudre le CPP sur G de manière optimale. Pour cela, on s'intéressera aux propriétés du graphe qui nous permettront de trouver une solution optimale.

3.1.1 Graphes Eulériens

Un graphe $G = (S, U)$ est dit Eulérien si on peut trouver un cycle qui passe une seule fois par chaque arête de G. Une condition équivalente est que le nombre de sommets de degré impair soit 0 ou 2 [21]. Dans le cadre du CPP, si le graphe est Eulérien, ce cycle sera donc la solution optimale du problème. En effet, le cycle passe une seule fois par arête donc on ne peut pas trouver un cycle passant par chaque arrête de longueur strictement inférieure.

Le principe de résolution du problème consistera donc à transformer le graphe de départ en graphe Eulérien en lui ajoutant des arêtes déjà existantes afin que tous ses sommets aient un degré pair, tout en minimisant le coût des arêtes ajoutées. On pourra ensuite utiliser l'algorithme d'Euler-Hierholzer pour trouver un chemin Eulérien dans le graphe modifié. On sait que le nombre de sommets de degré impair est pair [22]. Par la suite, on notera S' l'ensemble de sommets de degré impair.

3.1.2 Couplage parfait de poids minimum

Pour rendre le graphe Eulérien, il faut donc ajouter des arêtes entre les sommets de degré impair pour rendre leur degré pair. Tous ces sommets ne sont pas connectés entre eux, mais ajouter des arêtes de long d'un chemin qui les relie permet de les rendre pair sans pour autant changer le degré des nœuds traversés par ce chemin. On cherche en plus à minimiser le poids des arêtes ajoutées. Comme $|S'|$ est pair, on pourra relier des couples de S' et ainsi les transformer en sommet de degré pair, jusqu'à qu'il n'y ait plus de sommet de degré impair disponible. Pour simplifier le problème, on construira un graphe $G' = (S', U')$ à partir des sommets de degré impair où le poids sur chaque arête correspondra à la distance minimale entre ses deux extrémités dans G. Trouver quels chemins ajouter reviens à chercher un couplage parfait dans G'. De plus, on cherche à minimiser le coût des arêtes ajoutées, le couplage parfait devra donc être de poids minimum.

Une fois ce couplage trouvé, il faudra ajouter les arêtes correspondantes au couplage dans G. Le nouveau graphe G'' ainsi construit sera Eulérien. Le Blossom algorithm [6][15] permet de résoudre un problème de couplage maximum de poids maximum dans un graphe quelconque. Nous expliquerons dans la sous-section suivante comment cet algorithme nous permet également de trouver un couplage parfait de poids minimum dans notre cas.

FIGURE 4 – Résolution d'un problème de couplage parfait de poids minimum

3.1.3 Algorithme de résolution

La résolution du CPP consiste à :

1. Extraire S' l'ensemble des sommets de degré impair de G
   → **Complexité** : $o(|S|)$
2. Construire le graphe $G' = (S', U')$ et calculer le poids des arêtes de U' ($|S'|$ fois Dijkstra)
   → **Complexité** : $o(|S'|^2 + |S'||U'|)$
3. Trouver un couplage parfait de poids minimum sur G' (Blossom algorithm)
   → **Complexité** : $o(|S'|^2|U'|)$
4. Doubler les arêtes de G se trouvant dans le couplage pour créer G'' Eulérien
   → **Complexité** : $o(|U'|)$
5. Trouver un chemin Eulérien sur G'' (algorithme d'Euler-Hierholzer)
   → **Complexité** : $o(|U'|)$

La complexité de l'algorithme de résolution est en $o(|S'|^2|U'|)$.

Exemple On se donne un graphe $G = (S, U)$. La liste des sommets de degré impair est $S' = \{1, 2, 3, 4\}$. Par l'algorithme de Dijkstra, on trouve les plus courtes distances dans G entre les sommets de S' et on construit un graphe complet G' sur S' dont les arêtes ont le poids correspondant aux distances minimales entre les sommets dans G. Grâce à l'algorithme Blossom, on obtient le couplage parfait de poids minimum suivant $(1, 2)$ et $(3, 4)$ qui correspond aux arêtes $(1, 7)$, $(7, 2)$ et $(3, 4)$ dans G. On rajoute double ces arêtes dans G de façon à rendre G Eulérien. L'algorithme d'Euler-Hierholzer nous donne sur G le chemin Eulérien $6, 2, 3, 7, 2, 7, 1, 7, 4, 5, 1, 6$. La longueur totale du cycle optimal est 38.

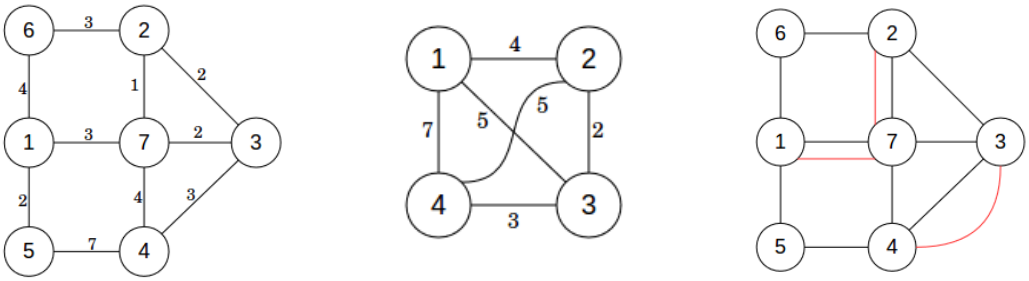


FIGURE 5 – Graphe G (gauche), graphe G' (centre) et graphe G'' Eulérien (droite)

3.2 Blossom algorithm

Le Blossom algorithm (ou algorithme des pétales) a été développé par Jack Edmonds en 1965. Il permet de résoudre un problème de couplage maximum dans un graphe non orienté connexe. Il tire son nom des cycles dans les graphes qui sont essentiels dans cet algorithme et qui ressemblent à des pétales. Par la suite, des versions améliorées de l'algorithme ont été développées pour améliorer la vitesse de l'algorithme ou pour résoudre des problèmes similaires. Avec cet algorithme, Edmonds a prouvé qu'un problème de couplage maximum était P dur.

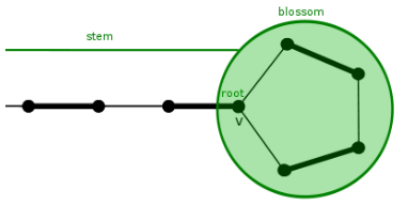


FIGURE 6 – Couplage (arêtes épaisses) et cycle faisant penser à une fleur

3.2.1 Couplage parfait de poids minimum

Soit $G = (S, U)$ un graphe non-orienté, un couplage K sur G est un sous-ensemble d'arêtes de U tel les arêtes de K ne soient pas adjacentes deux à deux. Il existe plusieurs problèmes de couplage avec différentes contraintes, mais pour le CPP, nous détaillerons le problème de couplage maximum et le problème de couplage de poids maximum. Un couplage est maximum s'il est de cardinalité maximale. Si K est un couplage maximum, on ne peut pas ajouter d'arête à K tel que K reste un couplage. Si on associe à chaque arête de U un coût $c(u)$, un couplage K sera de poids maximum s'il maximise $\sum_{u \in K} c(u)$. Un couplage K sur G est parfait si chaque sommet de G est adjacent à une unique arête de K.

Pour le CPP, il faut de trouver un couplage parfait de poids minimum sur l'ensemble S' des sommets de degré impair. Il a été vu précédemment que le nombre de sommets dans S' est pair. De plus, le graphe G' construit à partir de S' est un graphe complet. Trouver un couplage parfait revient donc à trouver un couplage maximum sur G'. Trouver un couplage de poids minimum est équivalent à un problème de couplage maximum dans un graphe où les signes des poids sur les arêtes aurait été changés. Le problème de couplage parfait de poids minimum est équivalent à un problème de couplage maximal de poids maximum pour un graphe avec un nombre de sommets pair au changement près du signe des poids.

Démonstration Un couplage maximum sur S' a une cardinalité inférieure ou égale à $\frac{S'}{2}$. Si le couplage est parfait, sa cardinalité est égale à $\frac{S'}{2}$. Tous les sommets de S' sont reliés entre eux. Supposons que K soit un couplage maximum sur S' tel que $|K| < \frac{S'}{2}$, alors il existe au moins deux sommets de S' n'ayant pas d'arêtes de K adjacentes. On peut alors ajouter l'arête (qui existe) entre ces sommets à K pour augmenter sa cardinalité d'où K n'est pas maximal. Si K est maximal, on a donc $|K| = \frac{S'}{2}$ et donc K est parfait. Le couplage qu'on cherche doit être de poids minimum. En changeant les signes des coûts des arêtes, ceci revient à trouver un couplage de poids maximum. En effet, pour $c(u) > 0 \quad \forall u \in U$, si K maximise $\sum_{u \in K} -c(u)$, il minimise $\sum_{u \in K} c(u)$.

3.2.2 Problème de couplage maximum

Soit $G = (S, U)$ un graphe et K un couplage sur ce graphe.

Saturation : Soit $s \in S$, on dit que s est saturé si $s \in K$, sinon, il est insaturé.

Chaîne alternée : Une chaîne alternée est un chemin C dans G telle que les arêtes de C soient alternativement dans K et dans $\bar{K} = U - K$. On conviendra que les arêtes dans K sont dessinées en traits épais et les arêtes dans $\bar{K}$ sont dessinées en traits fins. Soit L une chaîne alternée telle que les extrémités de L soient insaturées, alors le couplage K' obtenu en échangeant les arêtes fines et épaisses est appelé transfert le long de L. Cette transformation (appelée augmentation) augmente strictement la cardinalité du couplage, on appellera une telle chaîne une chaîne augmentante.

Cycle pair et impair : Un cycle élémentaire de G est pair (resp. impair) s'il content un nombre pair (resp. impair) de sommet. On appellera graphe contracté un graphe dans lequel les cycles élémentaires sont remplacés par un unique sommet en conservant les arêtes en périphérie du cycle.

Arbre alterné : Un arbre alterné de racine i_0 sur un couplage K est un arbre de G, de racine i_0 tel que i_0 soit le seul sommet insaturé de l'arbre et que les arêtes le long des branches de l'arbres soient alternativement dans K et $\bar{K}$.

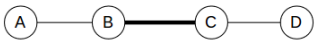


FIGURE 7 – Chaîne alternée

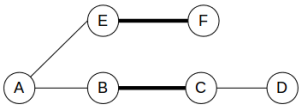


FIGURE 8 – Arbre alterné de racine A

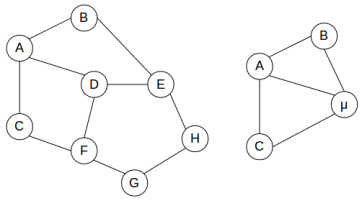


FIGURE 9 – Cycle impair $DEFGH$ et son graphe contracté μ

Théorème (Berge 1957) : Un couplage K est maximum s'il n'existe pas de chaîne alternée augmentante relativement à K.

Lemme : Soit Y l'ensemble des sommets d'un cycle élémentaire impair μ de G. Si K_1 est un couplage de $G - Y$, alors il existe un couplage maximum K_Y de μ tel que $K = K_1 \cup K_Y$ soit un couplage de G.

En trouvant une chaîne alternée ayant une extrémité insaturée et un cycle impair à l'autre extrémité, on pourra réaliser une augmentation le long de cette chaîne qui augmentera strictement le nombre de sommets dans le couplage. Le principe de l'algorithme sera de construire premier couplage arbitraire puis d'améliorer ce couplage en trouvant des chaînes augmentantes. Comme on améliore strictement le couplage en augmentant une chaîne, il faudra au plus $\frac{|S|}{2}$ augmentations pour avoir un couplage maximum. Les cycles impairs seront contractés au fur et à mesure de l'avancement de l'algorithme afin de simplifier la construction de l'arbre alterné. Ceci n'aura pas d'effet sur le couplage construit. À la fin de l'algorithme, on pourra éclater les cycles contractés et on devra effectuer un transfert sur le cycle pour le rendre compatible [23].

3.2.3 Problème de couplage de poids maximum

Soit $G = (S, U)$ un graphe où chaque arête $u \in U$ est munie d'un coût $c(u)$. On peut remarquer que ce couplage n'est pas forcément un couplage maximum.

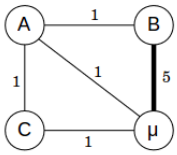


FIGURE 10 – Couplage de poids maximum qui n'est pas un couplage maximum

Coût réduit : Soit L une chaîne alternée, K un couplage sur G et $c(A)$ le coût total des arêtes contenues dans un ensemble A, le coût réduit $\delta(L)$ de L est $\delta(L) = c(L \cap \bar{K}) - c(L \cap K)$. Si K' est un couplage obtenu par transfert le long de L sur K, alors $\delta(L) = c(K') - c(K)$.

Théorème (White 1967) : Soit K un couplage de cardinalité p de poids maximum, soit L une chaîne alternée de coût réduit maximum. Alors le couplage K' obtenu à partir de K par transfert le long de L est de poids maximum parmi les couplages de cardinalité $p + 1$.

Théorème : Un couplage K est de poids maximum si et seulement s'il n'existe pas de chaîne alternées ni de cycle alterné pair de coût réduit strictement positif.

Le principe de résolution du problème de couplage de poids maximum sera le même que pour celui du problème de couplage maximum. Il faudra simplement sélectionner la chaîne alternée L et coût réduit minimal à chaque étape et s'arrêter lorsque toutes les chaînes alternées ont un coût réduit négatif.

3.2.4 Couplage maximum de poids maximum

Pour obtenir un couplage de poids maximum pour une cardinalité k fixée, il faudra non pas construire un arbre alterné, mais une forêt alternée de k arbres lors de la recherche du couplage de poids maximum [34]. Dans le CPP, on peut construire un couplage parfait sur un graphe $G' = (S', U')$ complet avec un nombre de sommets pair. En utilisant l'algorithme de recherche de couplage de poids maximum avec une forêt de $\frac{|S'|}{2}$ arbres, on trouvera un couplage maximum qui sera parfait.

3.3 Problème du postier Chinois orienté

Soit $G = (S, U)$ un graphe orienté, on cherche à résoudre le DCPP sur G. Comme pour le CPP, on s'intéressera aux propriétés du graphe et on le transformera de telle manière à pouvoir trouver un chemin Eulérien.

3.3.1 Graphes fortement connexe et degré des sommets

Un graphe G est fortement connexe si pour tout chemin $i \to j$ de G, il existe un chemin $j \to i$. Pour le DCPP, il n'existe pas toujours de solution. En effet, si le graphe n'est pas fortement connexe, il sera impossible de former un cycle. Par la suite, on considérera uniquement la famille des graphes orientés fortement connexes.

Un graphe Eulérien orienté possède uniquement des sommets qui ont le même nombre d'arêtes entrantes que sortantes. On note $d_+(s)$ (resp. $d_-(s)$) le nombre d'arêtes entrantes (resp. sortantes) dans s. Soit $S' = \{s \in S \mid d_+(s) \neq d_-(s)\}$, on sait que $\sum_{s \in S'} d_+(s) = \sum_{s \in S'} d_-(s)$ [20]. Notons

$S_1 = \{s \in S \mid d_+(s) > d_-(s)\}$ et $S_2 = \{s \in S \mid d_+(s) < d_-(s)\}$. Il faut trouver des chemins de poids minimum allant de S_2 à S_1 de telle manière que si on construit $G' = (S, U')$ en doublant ces chemins dans G, on ait $\forall s \in S,\ d_+(s) = d_-(s)$. Trouver les chemins de S_2 à S_1 revient à un problème d'affectation sur un graphe biparti. Ceci peut être résolu par l'algorithme de Hongrois avec un complexité en $o\big((|S_1| + |S_2|)^4\big)$. Ainsi, G' sera le graphe Eulérien de coût minimum pouvant être construit à partir de G. On pourra donc y trouver un chemin Eulérien qui sera la solution du DCPP.

Démonstration Soit $u \in U' - U$ allant de s_1 à s_2, si on retire u de G', alors le graphe n'est plus Eulérien. En effet, u n'a pas le même sommet pour source et destination, car $s_1 \in S_1$, $s_2 \in S_2$ et $S_1 \cup S_2 = \emptyset$. On aura diminué le degré sortant (resp. entrant) de s_1 (resp. s_2) d'une unité. On aura $d_+(s_1) \neq d_-(s_1)$ et $d_+(s_2) \neq d_-(s_2)$ donc le graphe ne sera pas Eulérien. Comme les arêtes ajoutées constituent un chemin de poids minimum, on ne peut pas construire de graphe Eulérien à partir de G ayant une longueur totale inférieure à G'.

3.3.2 Algorithme de résolution

La résolution du DCPP consiste à :

1. Extraire les ensembles $S_1 = \{s \in S \mid d_+(s) > d_-(s)\}$ et $S_2 = \{s \in S \mid d_+(s) < d_-(s)\}$
   → **Complexité** : $o(|S|)$
2. Résoudre un problème d'affectation simple (Hongrois)
   → **Complexité** : $o(|U|^4)$
3. Ajouter le nombre d'arêtes correspondant au chemins le plus court dans G aux solutions du problème d'affectation pour construire G' Eulérien
   → **Complexité** : $o(|U|)$
4. Trouver un chemin Eulérien sur G' (algorithme d'Euler-Hierholzer)
   → **Complexité** : $o(|U'|)$

La complexité de l'algorithme de résolution est en $o(|U|^4)$.

Exemple Soit $G = (S, U)$ un graphe. On extrait $S_2 = \{C\}$ et $S_1 = \{B\}$. Le problème d'affectation simple est ensuite trivial. On double les arêtes de G correspondant à la solution du problème d'affectation simple de telle sorte à construire G' Eulérien. L'algorithme d'Euler-Hierholzer nous donne ensuite la solution $A \to C \to B \to D \to B \to D \to C \to D \to C \to A$.

FIGURE 11 – Graphe G (gauche) et graphe Eulérien G' (droite) construit avec leurs poids

3.4 Problème du postier Chinois rural

Soit $G = (S, U)$ un graphe connexe non-orienté, on cherche à résoudre le RCPP pour un sous ensemble U' d'arêtes requises.

3.4.1 Complexité du problème

Ce problème est complètement dépendant du choix de l'ensemble U'. Notons G' le sous-graphe formé de U' et des sommets qui sont aux extrémités des arêtes de U'. Si G' est fortement connexe, ce problème revient à un problème du postier Chinois dans G'. Lorsque G' forme un graphe non-connexe, le problème devient NP complet. Il existe un algorithme de Branch-and-Bound basé sur les arbres de poids minimum optimal pour résoudre ce problème [4] en un temps non-polynomial. Lorsque la taille du graphe augmente, on se tournera plutôt vers des résolutions approchées par heuristiques.

3.4.2 Heuristique

L'algorithme de résolution heuristique présenté ici a été proposée par W.L. Pearn et T.C. Wu [32]. Il comporte 3 phases. On note G_R le sous graphe contenant les arêtes requises. La première consiste à transformer le graphe G_R en un graphe complet, la seconde consiste à effectuer une recherche d'arbre couvrant de poids minimum pour rendre G_R connexe et la dernière étape consiste à résoudre un problème du postier Chinois dans G_R dans lequel on a ajouté les arêtes de l'arbre couvrant de poids minimum.

Phase 1 - Transformation du graphe : Soit $G_R = (S_R, U_R)$ le sous graphe de G contenant les arêtes requises. On transforme G_R en graphe complet. Soit U_A les arêtes artificielles ajoutées pour rendre G_R complet. Le coût d'une arête de U_A allant de i à j correspond au coût minimum pour aller de i à j dans G. Le graphe ainsi construit est noté $G_{RC_0} = (S_R, U_R \cup U_A)$. On simplifie G_{RC_0} en retirant i) les arêtes $(i, j) \in U_A$ telle que $\exists k,\ c_{ij} = c_{ik} + c_{kj}$ et ii) une des deux arêtes si deux arêtes parallèles ont le même coût. Le graphe obtenu est noté G_{RC}

Phase 2 - Arbre couvrant de poids minimum : Soit $\{C_1, ..., C_r\}$ les composantes de G_R et G_C le graphe condensé de G où chaque $C_i,\ i \in 1, .., r$ est traité comme un unique sommet. Une arête (i, j) existe dans G_C si il existe une arête $(x, y) \in G_{RC}$ telle que $x \in C_i,\ y \in C_j$. Le poids de l'arête (i, j) sera alors $d(i, j) = \min_{x,y}\{d(x, y) - u_x - u_y\}$. Avec $u_i = \eta(deg(i) - 2)$, où $deg(i)$ est le degré du sommet i, η est le paramètre de l'heuristique. On cherche un arbre couvrant de poids minimum sur G_C et on note U_T les arêtes de cet arbre.

Phase 3 - Problème du postier Chinois : On résout un problème du postier Chinois dans $G_R \cup E_T$ et on obtient un cycle Eulérien. Pour résoudre ce problème, on a dû trouver un couplage parfait de poids minimum donc on note les arêtes S_M. Alors, la solution de notre problème sera le chemin Eulérien dans $G_R \cup U_T \cup U_M$.

3.4.3 Exemple

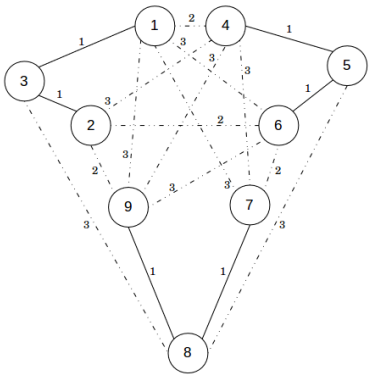


FIGURE 12 – Graphe de base où les arêtes non-requises sont en pointillé

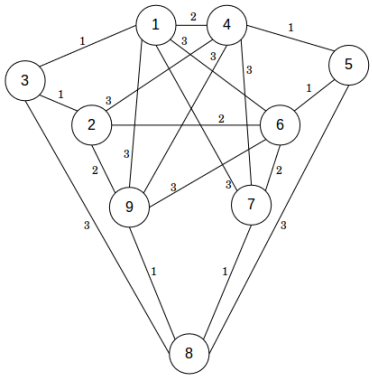


FIGURE 13 – Graphe après transformation (étape 1)

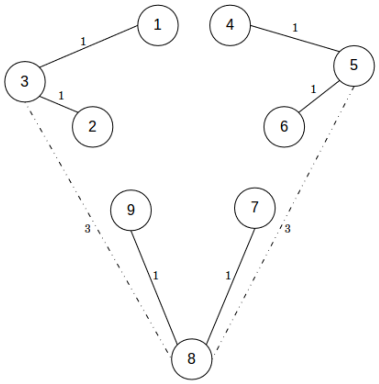


FIGURE 14 – Arbre couvrant de poids minimum (étape 2)

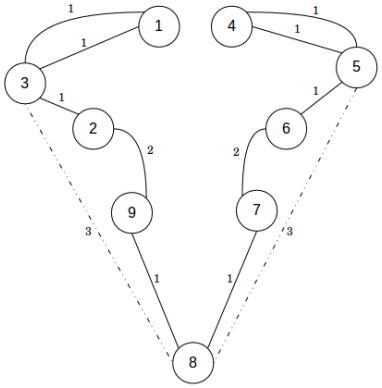


FIGURE 15 – Résolution du problème du postier Chinois (étape 3)

Le chemin trouvé par la méthode heuristique est $1, 3, 2, 9, 8, 7, 6, 5, 4, 5, 8, 3, 1$ et il est de longueur 18. Le chemin optimal de ce problème est $1, 3, 2, 9, 8, 7, 6, 5, 4, 1$ et est de longueur 12.

4 Programmation Linéaire

Pour résoudre le GOP, nous allons utiliser des méthodes des programmation linéaire. Cette section résume quelques techniques de résolution et représente les recherches bibliographiques effectuées autour de ce sujet.

4.1 Définitions

Un programme linéaire (PL ou PLs lorsque le nombre de variables/contraintes peut être exponentiel) consiste à maximiser/minimiser une fonction objectif sous des contraintes s'exprimant sous forme d'équations ou d'inéquations linéaires. Développée à partir des années 1940 notamment par George Dantzig, cette approche permet de modéliser et de résoudre beaucoup de problèmes d'optimisation. Ces problèmes peuvent être exprimés sous différentes formes et plusieurs algorithmes permettent de les résoudre.

4.1.1 Forme des PLs

Forme générale
Un PLs est sous forme générale si on exprime ses contraintes à l'aide d'équations et d'inéquations linéaires. La forme sera la suivante :

$$\begin{aligned}
\text{Maximize : } & z = \sum_j c_j x_j \\
\text{Subject to : } & \sum_j a_{ij} x_j \geq b_i \qquad \forall i \in I_+ \\
& \sum_j a_{ij} x_j \leq b_i \qquad \forall i \in I_- \\
& \sum_j a_{ij} x_j = b_i \qquad \forall i \in I_0 \\
& x_j \geq 0 \qquad \forall j \in 1, .., m
\end{aligned}$$

Avec $I_+,\ I_-,\ I_0$ disjoints et $m, n, a_{ij}, b_i, I_+, I_-, I_0$ connus. La restriction des x_j à $\mathbb{R}_+$ se fait sans perte de généralité car on peut exprimer tout $x \in \mathbb{R}$ comme $x = x_1 - x_2$, où $x_1, x_2 \in \mathbb{R}_+$.

Forme standard
La forme standard d'un PLs s'exprime avec des contraintes qui sont uniquement des équations linéaires. La forme sera la suivante :

$$\begin{aligned}
\text{Maximize : } & z = c^T x \\
\text{Subject to : } & Ax = b \\
& x \succeq 0
\end{aligned}$$

Avec $x \in \mathbb{R}^n_+, c \in \mathbb{R}^n, b \in \mathbb{R}^m$ et A une matrice de taille $m \times n$ et de rang m où A, b, c, n et m sont connus. Il est toujours possible de transformer un PLs sous forme générale en PLs sous forme standard en ajoutant des variables non-significatives pour le problème, appelées variables d'écart. En effet, $ax \leq b \iff ax + u = b, \quad u \geq 0$.

4.1.2 Ensemble de solutions

L'ensemble qui vérifie les contraintes d'une PLs sous forme générale est appelé ensemble réalisable. Il est défini par un polyèdre convexe dans $\mathbb{R}^n_+$. On peut montrer que si la solution du PLs est unique, elle se situe sur un sommet du polyèdre, sinon il existe une infinité de solutions qui constituent une arête du polyèdre [28].

$$\text{Maximize} : z = x + y$$
$$\text{Subject to} : \begin{pmatrix} 15 & 1 \\ -1 & 1 \\ 10 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \preceq \begin{pmatrix} 4 \\ 2 \\ 50 \end{pmatrix}$$
$$\begin{pmatrix} x \\ y \end{pmatrix} \succeq 0$$

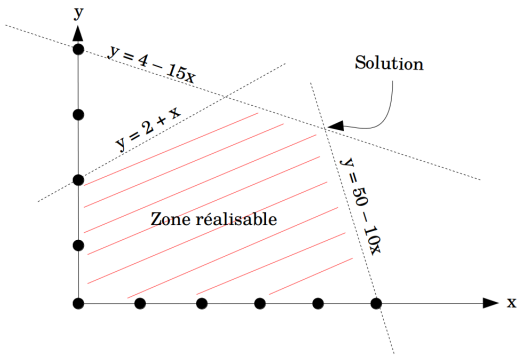


FIGURE 16 – Exemple de PLs avec son polyèdre associé

4.2 Résolution

Une manière naïve de trouver une solution serait d'évaluer la fonction objectif sur tous les sommets du polyèdre. Comme la solution se trouve sur un sommet, le point solution serait celui où l'évaluation de la fonction objectif est le plus grand. Dans le cas où deux sommets donnent la même évaluation de la fonction objectif, la solution serait alors l'ensemble des points contenus dans l'arête du polyèdre joignant les deux sommets. Néanmoins, le nombre de sommets dans le polyèdre croît dans de nombreux problèmes de manière exponentielle par rapport au nombre de variables et de contraintes. Énumérer tous les sommets n'est donc souvent pas possible en pratique. C'est pour cela que plusieurs méthodes qui permettent de converger vers la solution optimale du PLs ont été développées au cours du XXème siècle.

Le premier algorithme efficace inventé pour résoudre un PLs est l'algorithme du simplexe. Son invention est due à Georges Dantzig en 1947. Le principe est de se déplacer de sommet en sommet sur le polyèdre de telle manière à toujours faire augmenter la fonction objectif. Si on ne peut plus faire augmenter la fonction objectif, on a alors trouvé la solution optimale. Si on peut avancer infiniment loin sur une arête du polyèdre de telle manière à toujours faire augmenter la fonction objectif, alors le PLs est non borné. Cet algorithme est très efficace en moyenne et est encore très souvent utilisé dans les résolutions de PLs. Cependant, en 1972, Klee et Minty ont montré que dans certains cas, l'algorithme devait effectuer un nombre exponentiel d'itérations avant de converger [33].

La méthode dite ellipsoïde a été développée par Kachian en 1979. Elle se montre plus efficace que la méthode du simplexe sur certains exemples, mais pas en moyenne. En 1984, Karmarkar a été le premier à exhiber un algorithme polynomial permettant de résoudre un PLs. Cet algorithme est basé sur la méthode des points intérieurs [13]. Il sera récompensé du prix Fulkerson en 1988 pour cela. Des algorithmes comme l'algorithme de barrière [5] ont été développés à partir de l'algorithme de Karmarkar et sont utilisés aujourd'hui dans les différents solvers.

4.3 PLs en grande dimension

La plupart du temps, le nombre de variables et de contraintes introduites pour résoudre le problème est très importantes. Dans ce ca, il faut introduire de nouveaux algorithmes qui, couplés à l'algorithme du simplexe, pourront accélérer la résolution du PLs. Ces algorithmes sur les particularités de la matrice de contraintes A du PLs.

4.3.1 Génération de colonnes

Dans certains problèmes, la nombre de variables n est très supérieur au nombre de contraintes m. L'algorithme de génération de colonnes repose sur le fait que le nombre de variables non-nulles à l'optimum n'excède jamais le nombre de contraintes [26]. Le principe sera d'exhiber itérativement les variables qui seront non nulles à l'optimum. Cette technique est très intéressante lorsque le nombre de variables est exponentiel, mais le nombre de contraintes est polynomial.

On appelle base toute sous-matrice de taille $m \times m$ de A. Ainsi, si B est une base, on pourra exprimer A comme deux sous-matrices (B et N) et on pourra séparer le vecteur x en deux parties associées chacune à B ou N. On pourra donc écrire :

$$Ax = b \Longleftrightarrow (B|N).(x_B \ x_N)^T = b$$

Pour appliquer l'algorithme de génération de colonnes, on doit supposer l'existence d'un algorithme efficace (polynomial) permettant de trouver la colonne s de A qui minimise $z(A_s) = c_s - \pi.A_s$, où $\pi \in \mathbb{R}^m$. On partira d'une base réalisable, mais pas forcément optimale puis, grâce à l'algorithme efficace d'extraction de colonne, on fera rentrer dans la base la nouvelle variable correspondant à la colonne extraite de A. Ainsi, on résoudra à chaque itération un PLs de taille nettement inférieur. On pourra donc utiliser la méthode du simplexe. Lorsqu'on doit ajouter une colonne s telle que $c_s - \pi.A_s > 0$, on a trouvé les variables qui seront non nulles (actives) à l'optimum et on aura réduit le nombre de variables et donc la dimension du PLs.

4.3.2 Décomposition de Dantzig

Dans la plupart des modélisations de PLs, la matrice A est fortement creuse. De plus, les "0" de A ne sont pas disposés de manière aléatoire, ils forment souvent des blocs, car certaines variables sont liées à une partie seulement des contraintes. Un cas classique est le suivant : un certain nombre de contraintes lient les variables entre elles et le reste des contraintes forment des problèmes indépendants dans lesquels est inclut un certain sous-ensemble des variables.

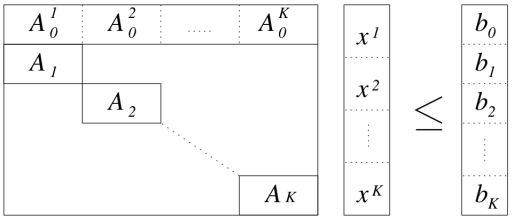


FIGURE 17 – Décomposition de A en blocs non-creux

Les parties vides correspondent à des blocs de "0" dans A. La première ligne de blocs correspond aux contraintes couplantes (qui lient toutes les variables entre elles) et les autres blocs forment des sous problèmes indépendants les uns des autres. On peut donc reformuler le PLs :

$$\begin{aligned}
&\text{Maximize : } z = c^T.x \\
&\text{Subject to : } \sum_{k=1}^{K} A_0^k x^k \leq b_0 \\
&\qquad\qquad A_k.x^k \leq b_k, \qquad \forall k \ \in \ 1, ..., K \\
&\qquad\qquad x_k \geq 0, \qquad \forall k \ \in \ 0, ..., K
\end{aligned}$$

Le principe de la décomposition de Dantzig est d'introduire la fonction duale du problème ce qui aura pour effet de séparer les contraintes couplantes en $K+1$ sous-problèmes indépendants de dimension réduite qu'on pourra alors résoudre grâce à l'algorithme du simplexe. On associe à chaque contrainte A_0^i un multiplicateur de Lagrange λ_i. Le vecteur λ regroupera ces multiplicateurs. On peut écrire la fonction de Lagrange et la fonction duale du PLs de la manière suivante :

$$\mathcal{L}(x,\lambda) = c^T.x - \lambda(Ax-b) = \lambda b + \sum_{k=0}^{K}(c_k - \lambda A_k).x_k$$

$$\omega(\lambda) = \min_x \Big\{\mathcal{L}(x,\lambda)\Big\} = \lambda.b + \sum_{k=0}^{K} \min_{x_k} \Big\{(c_k - \lambda A_k).x_k\Big\}$$

À l'aide de la fonction duale, on a donc décomposé le problème en $K+1$ sous-problèmes indépendants qui pour un k fixé ont la forme suivante :

$$\begin{aligned} \text{Maximize : } & z_k = (c_k - \lambda A_k).x_k \\ \text{Subject to : } & A_k.x^k \leq b_k \\ & x_k \geq 0 \end{aligned}$$

Ceux-ci pourront être résolus avec l'algorithme du simplexe et on concaténera les x_k solutions des sous-problèmes pour avoir la solution du PLs général.

M. Minoux nous donne une interprétation économique intéressante de cette décomposition [27]. On considère une grande entreprise constituée de K filiales indépendantes, avec chacune un niveau d'activité x_k, qui fonctionnent indépendamment les unes des autres. On suppose qu'elles doivent toutes maximiser leur profit $z_k = c_k x_k$ avec des contraintes données par A_k. Les ressources nécessaires au fonctionnement des filiales sont néanmoins regroupées de manière global dans l'entreprise. Les quantités b_k sont les quantités des différentes ressources à se partager dans l'entreprise. L'entreprise décide de changer de fonctionnement et de faire payer un certain prix λ_k chacune des ressources attribuées aux filiales. Les filiales devront donc ajuster leur fonction à maximiser de telle manière à prendre en compte ce nouveaux prix à payer qui rendra compte de l'impact de l'achat d'une ressource au niveau global de l'entreprise (une ressource achetée par une filiale est peut être vitale pour le fonctionnement d'une autre filiale). La nouvelle fonction à maximiser sera alors $\tilde{z} = c_k x_k - \lambda_k A_k x_k$.

4.4 Adaptation aux PLNE

4.4.1 Discrétisation

Dans certains PLs, on cherche une solution entière. On appelle cela un programme linéaire en nombres entiers (PLNE). La plupart du temps, il est impossible d'énumérer tous les points à coordonnées entières dans la zone réalisable car le nombre de facettes du polyèdre de l'espace réalisable croît souvent de manière exponentielle en fonction du nombre de variables et de contraintes. Il est également impossible d'utiliser des méthodes classiques d'optimisation, qui se basent souvent sur le principe de pente et de dérivées, de part la nature du problème. Afin de trouver quel point à coordonnées entières est la solution du PLNE, on peut appliquer différentes méthodes comme le Branch-and-Bound ou les méthodes de coupe.

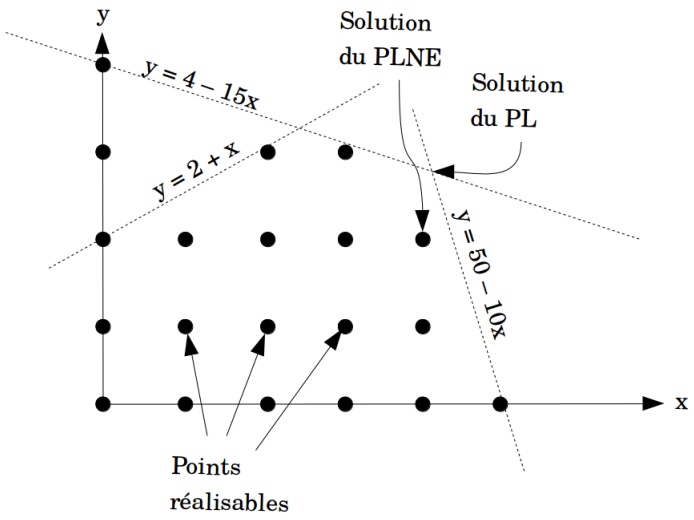


FIGURE 18 – Solution du PLNE et sa solution relaxée (solution sans les contraintes d'intégrité)

4.4.2 Branch-and-Bound

Le Branch-and-Bound consiste à énumérer les solutions de manière efficace, c'est-à-dire de choisir les quelles sont intéressantes à évaluer et lesquelles ne le sont pas. On représente ces choix sous forme d'un arbre. Le Branch-and-Bound consiste à parcourir cet arbre et à chaque étape choisir quelle branche (Branch) on va explorer. Pour cela, on résout une relaxation linéaire du problème qui nous donnera une borne (Bound) aux solutions entières. Si solution $\bar{x}$ de la relaxation est entière, alors c'est une potentielle solution au PLNE. Sinon, on sépare l'arbre en deux branches : une où $x_i \geq \lfloor \bar{x_i} \rfloor$ et une où $x_i \leq \lceil \bar{x_i} \rceil$ puis on choisi la branche la plus prometteuse. Lorsqu'une relaxation linéaire donne une solution relaxée qui ne permet pas d'améliorer la meilleure solution, on arrête l'exploration dans la branche (Cut). La solution du PLNE sera la meilleure solution entière trouvée lorsqu'il ne restera plus de branches à explorer.

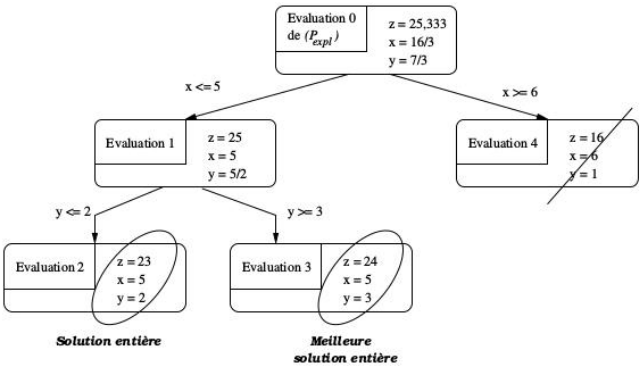


FIGURE 19 – Application de la méthode branch-and-bound avec z la fonction objectif à maximiser et x,y les coordonnées de la variable

4.4.3 Méthodes de coupe

Les méthodes de coupe consistent à trouver l'enveloppe convexe des solutions à coordonnées entières. Une fois cette enveloppe trouvée, on peut utiliser l'algorithme du simplexe en étant certain que la solution trouvée aura des coordonnées entières. Les premières méthodes de coupes ont été introduites par Gomory en 1958 [12]. On résout le problème relaxé puis on introduit une nouvelle contrainte permettant de réduire la zone réalisable sans exclure de solutions à coordonnées entières puis on réitère l'opération sur la nouvelle zone de recherche. Il faut bien connaître la structure du problème et ses caractéristiques pour pouvoir créer des coupes intéressantes, sinon, la recherche de coupe peut mettre plus de temps que la résolution en nombre entiers. Les algorithmes récents combinent des méthodes de coupes et de Branch-and-Bound.

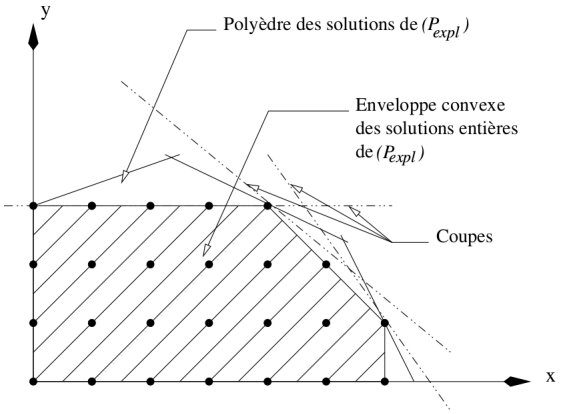


FIGURE 20 – Méthode de coupe

4.4.4 Branch-and-Price

La méthode de Branch-and-Price est une méthode hybride de résolution de PLNE qui couple méthode de génération de colonnes et Branch-and-Bound. Elle est utile lorsqu'on peut séparer le problème initial en sous-problèmes. Dans ce cas-là, on applique la méthode de la décomposition de Dantzig pour construire un problème maître (contraintes couplantes) et des problèmes esclaves indépendants (sous-problèmes indépendants). Le problème maître réduit correspond au problème maître dans lequel on ne considère qu'un sous-ensemble des colonnes du problème maître. On va parcourir un arbre dans lequel on résoudra des problèmes maître et esclaves et à chaque branchement, on rajoutera des contraintes qui permettront de séparer l'espace de recherche en deux de telle manière à converger vers une solution entière.

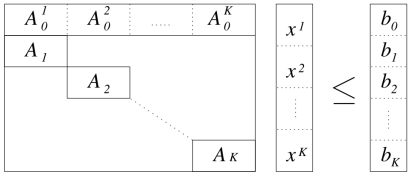


FIGURE 21 – Décomposition de A un problème maître (première ligne de A) et des problèmes esclaves indépendants (blocs $A_i,\ i \in \{1, ..., K\}$)

On associe à chaque problème esclave un coût réduit qui correspond à la valeur qu'on doit ajouter à la fonction objectif du problème maître pour que la prise en compte du problème esclave dans le problème maître réduit soit bénéfique pour la résolution. Tant qu'on peut trouver des problèmes esclaves avec un coût réduit négatif (dans le cas d'une minimisation), on les fait entrer dans le problème maître réduit puis on résout une relaxation de ce nouveau problème. Les relaxations nous donnent des bornes inférieures (pour une minimisation) aux solutions du problème maître réduit. Lorsqu'on ne peut plus trouver de solutions des problèmes esclaves avec un coût réduit négatif, on regarde si la solution obtenue lors de la dernière relaxation est entière. Si c'est le cas, alors cette solution est une borne supérieure aux solutions du problème. Si cette borne supérieure améliore la borne précédemment trouvée, on la met à jour. Sinon, on doit choisir une variable de branchement et créer une nouvelle branche dans l'arbre. On introduira des contraintes différentes dans les deux nœuds fils créés. Lorsque dans une branche, les solutions des relaxations donnent une borne inférieure qui est plus grande que la borne supérieure globale, on arête l'exploration de la branche, car il sera impossible d'obtenir de meilleures solutions.

Le but de résoudre une relaxation du problème maître est d'explorer vite l'arbre de branchements. Les problèmes esclaves doivent être résolus de manière entière donc on doit s'assurer que leur résolution ne prend pas trop de temps. L'efficacité de l'algorithme de Branch-abnd-Price repose sur la rapidité de résolution des problèmes maître et esclaves ainsi que sur le choix du branchement.

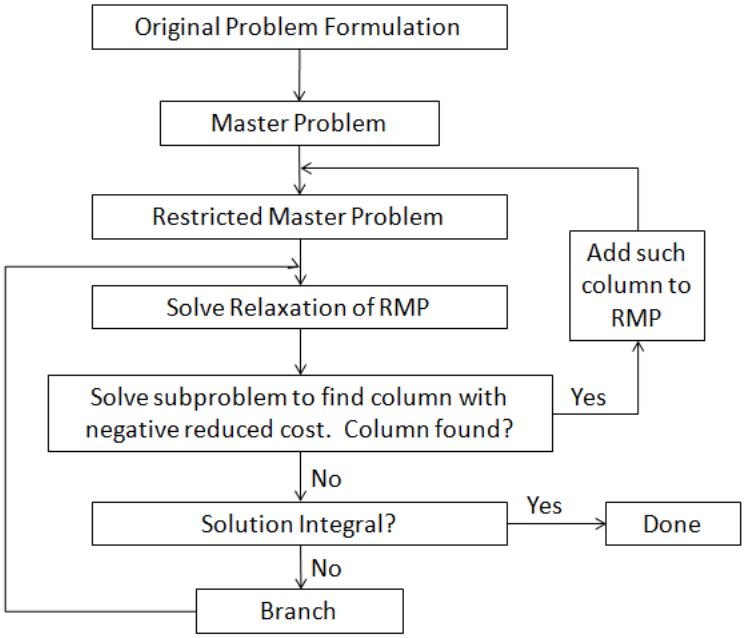


FIGURE 22 – Algorithme de Branch-and-Price

5 Formulations des PLNE pour un unique dépôt

5.1 Notations

Les notations seront similaires pour les différentes formulations mais elles ne seront pas forcément toutes utilisées. On note :

- G le graphe du problème (non-orienté, orienté ou mixte)
- S les sommets de G
- A' l'ensemble des arcs de G
- A_R l'ensemble des arcs requis de G
- E l'ensemble des arêtes de G
- E_R l'ensemble des arêtes requises de G
- $A = A' \cup \{(i,j),(i,j) \in E\} \cup \{(j,i),(i,j) \in E\}$ l'ensemble des arcs et des arêtes de G où chaque arête a été remplacée par deux arcs opposés (en conservant les caractéristiques de l'arête)
- $\Gamma = (S, A)$ le graphe orienté formé à partir de A
- R l'ensemble des arcs requis dans Γ ($|R| = 2|E_R| + |A_R|$)
- q_{ij} la demande sur l'arc $(i,j) \in R$
- d_{ij} le coût de passage sur l'arc $(i,j) \in A$
- c_{ij} le coût de service sur l'arc $(i,j) \in R$
- Q_T la demande totale sur G
- W la capacité maximale de chaque véhicule
- 0 le nœud de Γ correspondant au dépôt
- P le nombre de véhicules disponibles

5.2 CPP

On a $P = 1,\ A = A_R = \emptyset$ et $E = E_R$ (un seul véhicule, pistes praticables dans les deux sens et qui doivent toutes êtres damées).

Variables

- $y_{ij},\ (i,j) \in E$ est le nombre de fois qu'on passe par l'arête (i,j) sans la servir
- $k_i,\ i \in S$

Le nombre de passages sur une arête est $y_{ij} + 1$.

Formulation

$$\text{Minimize :} \quad \sum_{(i,j)\in E} c_{ij} + \sum_{(i,j)\in E} y_{ij} d_{ij} \tag{1}$$

$$\text{Subject to :} \quad \sum_{(i,j)\in E} (1 + y_{ij}) = 2k_i \qquad \forall i \in S \tag{2}$$

$$y_{ij} \in \mathbb{N} \qquad \forall (i,j) \in E \tag{3}$$

$$k_i \in \mathbb{N} \qquad \forall i \in S \tag{4}$$

Commentaires

L'équation (1) traduit le fait qu'il faille minimiser les coûts de service sur toutes les arêtes (car on les sert toutes une unique fois) ainsi que les coûts de passage sur les arêtes sur lequel on passe sans service. La première somme de (1) est constante et n'est donc pas utile dans cette formulation, mais pour plus de clarté, on la conserve. Les équations (2) et (4) permettent d'être sûr qu'on forme un chemin continu. En effet, si on rentre dans un nœud i, on doit pouvoir en sortir donc le nombre d'arêtes adjacentes à un nœud est pair. Enfin, (3) nous permet d'avoir un nombre de passage entier par arête.

5.3 DCPP

On a $P = 1,\ E = E_R = \emptyset$ et $A = A_R$ (un seul véhicule, pistes praticables dans un seul sens et qui doivent toutes êtres damées).

Variables

- y_{ij}, $(i,j) \in A$ est le nombre de fois qu'on passe par l'arc (i,j) sans le servir

Le nombre de passages sur un arc est $y_{ij} + 1$.

Formulation

$$\text{Minimize :} \quad \sum_{(i,j)\in A} c_{ij} + \sum_{(i,j)\in A} y_{ij} d_{ij} \tag{1}$$

$$\text{Subject to :} \quad \sum_{(i,j)\in A} (1 + y_{ij}) = \sum_{(j,i)\in A} (1 + y_{ji}) \quad \forall i \in S \tag{2}$$

$$y_{ij} \in \mathbb{N} \qquad \forall (i,j) \in A \tag{3}$$

Commentaires

Comme précédemment, la première somme de (1) n'est pas utile, mais elle sert à clarifier les notations. On souhaite minimiser le coût de service sur tous les arcs (car on doit tous les servir un unique fois) et les coûts de passage sur les arcs empruntés sans service. L'équation (2) permet de s'assurer que le chemin construit est continu . En effet, on doit pourvoir sortir autant de fois que l'on entre dans chaque sommet de S pour former un chemin. Enfin, (3) permet d'avoir un nombre de passage entier par arc.

5.4 MCPP

On a $P = 1$, $E = E_R$ et $A' = A_R$ donc $R = A$ (un seul véhicule, pistes praticables dans un seul ou dans les deux sens et qui doivent toutes être damées).

Variables

- $x_{ij} = \begin{cases} 1 \text{ si on sert l'arc } (i,j) \in R \\ 0 \text{ sinon} \end{cases}$
- y_{ij}, $(i,j) \in A$ est le nombre de fois qu'on passe par l'arc (i,j) sans service

Le nombre de passages sur un arc est $x_{ij} + y_{ij}$.

Formulation

$$\text{Minimize :} \quad \sum_{(i,j)\in A} x_{ij} c_{ij} + \sum_{(i,j)\in A} y_{ij} d_{ij} \tag{1}$$

$$\text{Subject to :} \quad \sum_{(i,j)\in A} (x_{ij} + y_{ij}) = \sum_{(j,i)\in A} (x_{ji} + y_{ji}) \quad \forall i \in S \tag{2}$$

$$x_{ij} = 1 \qquad \forall (i,j) \in A_R \tag{3}$$

$$x_{ij} + x_{ji} = 1 \qquad \forall (i,j) \in E_R \tag{4}$$

$$y_{ij} \in \mathbb{N} \qquad \forall (i,j) \in A \tag{5}$$

Commentaires

L'équation (1) traduit le fait qu'on veuille minimiser les coûts de service sur les arcs servis et les coûts de passage sur les arcs empruntés sans service. (2) permet d'assurer la continuité des chemins formés comme expliqué précédemment. (3) nous assure que chaque arc requis est servi et (4) nous assure que chaque arête requise de E_R est servie une seule fois (on peut y passer dans les deux sens, mais en la servant qu'une seule fois). Enfin, (5) permet d'avoir un nombre de passage entier par arc.

5.5 MCARP

Variables

- $x_{ij}^p = \begin{cases} 1 \text{ si le véhicule } p \text{ sert l'arc } (i,j) \in R \\ 0 \text{ sinon} \end{cases}$
- y_{ij}^p, $(i,j) \in A$ est le nombre de fois que le véhicule p passe par l'arc (i,j) sans le servir

- $f_{ij}^p,\ (i,j) \in A$ est le flot sur l'arc $(i,j) \in A$ correspondant à la demande restante dans le chemin effectué par le véhicule p

Le nombre de fois que le véhicule p passe sur un arc est $x_{ij}^p + y_{ij}^p$.

Formulation

$$\text{Minimize :} \quad \sum_{p=1}^{P} \left[\sum_{(i,j)\in R} x_{ij}^p c_{ij} + \sum_{(i,j)\in A} y_{ij}^p d_{ij} \right] \tag{1}$$

$$\text{Subject to :} \quad \sum_{(i,j)\in R} x_{ij}^p + \sum_{(i,j)\in A} y_{ij}^p = \sum_{(j,i)\in R} x_{ji}^p + \sum_{(j,i)\in A} y_{ji}^p \qquad \forall i \in S \qquad \forall p \in \{1,...,P\} \tag{2}$$

$$\sum_{p=1}^{P} x_{ij}^p = 1 \qquad \forall (i,j) \in A_R \tag{3}$$

$$\sum_{p=1}^{P} (x_{ij}^p + x_{ji}^p) = 1 \qquad \forall (i,j) \in E_R \tag{4}$$

$$\sum_{(0,j)\in A} y_{0j}^p + \sum_{(0,j)\in R} x_{0j}^p \leq 1 \tag{5}$$

$$\sum_{(j,i)\in A} f_{ji}^p - \sum_{(i,j)\in A} f_{ij}^p = \sum_{(j,i)\in R} x_{ji}^p q_{ji} \qquad \forall i \in S \setminus 0 \qquad \forall p \in \{1,...,P\} \tag{6}$$

$$\sum_{(0,j)\in A} f_{0j}^p = \sum_{(i,j)\in R} x_{ij}^p q_{ij} \qquad \forall p \in \{1,...,P\} \tag{7}$$

$$\sum_{(i,0)\in A} f_{i0}^p = \sum_{(i,0)\in R} x_{i0}^p q_{i0} \qquad \forall p \in \{1,...,P\} \tag{8}$$

$$f_{ij}^p \leq W(x_{ij}^p + y_{ij}^p) \qquad \forall (i,j) \in A \qquad \forall p \in \{1,...,P\} \tag{9}$$

$$f_{ij}^p \geq 0 \qquad \forall (i,j) \in A, \qquad \forall p \in \{1,...,P\} \tag{10}$$

$$y_{ij}^p \in \mathbb{N} \qquad \forall (i,j) \in A, \qquad \forall p \in \{1,...,P\} \tag{11}$$

Commentaires

L'équation (1) traduit le fait qu'on veuille minimiser la somme du coût de service sur les arcs servis ainsi que le coût de passage sur les arcs sur lesquels on passe et ce pour tous les véhicules. (2) permet d'avoir des chemins continus pour chaque véhicule comme dans le cas du MCPP. L'équation (3) (resp. (4)) nous assure qu'au moins un véhicule sert chaque arc requis (resp. sert chaque arête requise dans un seul sens). (5)-(9) sont les contraintes liées au flot qui permettent d'éviter des sous-tours. Une explication est faite au paragraphe suivant. (9) permet de garantir le respect de la capacité de chaque véhicule (on peut prouver cela en sommant sur les acrs de A puis en utilisant (5) et (7), on obtient alors $\sum_{(i,j)\in R} q_{ij} x_{ij}^p \leq W$). Enfin, (10)-(11) permettent d'avoir un flot positif et un nombre de passage entier. On peut remarquer que (5) impose qu'un tour se termine dès lors qu'il repasse au dépôt. Il se peut alors qu'un véhicule soit utilisé pour un tout petit cycle. Dans ce cas, on peut augmenter le nombre de véhicules puis assigner un véhicule à plusieurs petites tournées, tout en respectant sa capacité.

Variables de flot

Les variables de flots permettent de supprimer les sous-tours et elles représentent la demande restante dans un chemin pour chaque arc. L'équation (7) permet d'initialiser le flot sur le premier arc du chemin

avec la valeur de la demande totale du chemin. Pour chaque arc, le flot sera le flot de l'arc précédent moins la demande de l'arc qu'on vient de traverser (6). Enfin, la contrainte (8) permet de vérifier que sur le dernier arc, le flot correspond bien à la demande de l'arc. Dans le cas d'un sous-tour, on n'aurait pas pu décrémenter le flot des demandes des arcs compris dans le sous-tour et le flot sur l'arc terminant dans le dépôt serait donc supérieur à la demande de l'arc, (8) ne serait donc pas vérifiée.

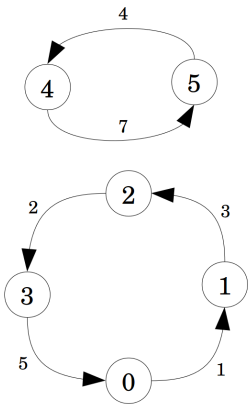


FIGURE 23 – Chemin avec un sous-tour (vérifiant quand même (2))

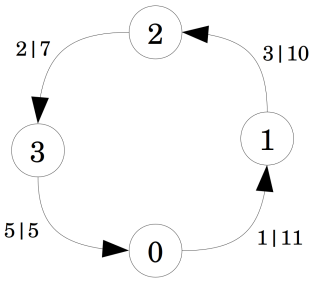


FIGURE 24 – Chemin valide avec demande et flot sur chaque arc

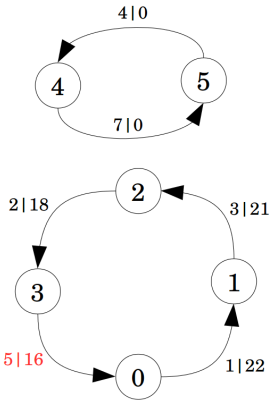


FIGURE 25 – Chemin non-valide avec demande et flot sur chaque arc

Contraintes valides supplémentaires pour accélérer la résolution

$$\sum_{p=1}^{P}\left(\sum_{(0,j)\in A} y_{0j}^{p} + \sum_{(0,j)\in R} x_{0j}^{p}\right) \geq \left\lceil \frac{Q_T}{W} \right\rceil \tag{12}$$

$$f_{ij}^{p} \geq x_{ij}^{p} q_{ij} \qquad \forall (i,j) \in R \qquad \forall p \in \{1,...,P\} \tag{13}$$

$$f_{ij}^{p} \geq y_{ij} - 1 \qquad \forall (i,j) \in A \setminus R \qquad \forall p \in \{1,...,P\} \tag{14}$$

$$\sum_{(0,j)\in A} y_{0j}^{p} + \sum_{(0,j)\in R} x_{0j}^{p} \geq \sum_{(0,j)\in A} y_{0j}^{p+1} + \sum_{(0,j)\in R} x_{0j}^{p+1} \qquad \forall p \in \{1,...,P-1\} \tag{15}$$

Les équations (12)-(15) ne sont pas nécessaires, mais elles permettent d'accélérer la résolution en fixant le nombre minimum de véhicules à avoir en fonction de leur capacité et de la demande totale pour (12), en imposant des bornes inférieures au flot pour (13)-(14) et en brisant les symétries l'association chemin/véhicule pour (15).

6 Branch-and-Price pour le cas avec plusieurs dépôts

Lorsque plusieurs dépôts sont disponibles on se trouve dans le cas du MD-MCARP. On peut avoir envie d'adapter la formulation du MCARP en ajoutant une indexation correspondant au dépôt associé au chemin de la dameuse. Malheureusement, les contraintes de flot sont difficilement adaptables pour ce cas. Il faudrait donc revenir sur une formulation avec un nombre exponentiel de contraintes. De plus, on devrait également introduire des variables d'association entre véhicule et dépôt qui seraient elles aussi en nombre exponentiel. Une modélisation et une résolution par solver prendrait beaucoup trop de temps. On propose donc un algorithme de Branch-and-Price pour résoudre ce problème. En effet, on peut facilement remarquer qu'on peut séparer le GOP en sous-problèmes traitants chacun d'une seule tournée associée à un seul dépôt. Au moment de l'écriture de ce rapport, il ne semble qu'aucun article de recherche ne traite ce problème de cette manière.

6.1 Notations supplémentaires

On reprend en partie les notations du MCARP et on introduit des notations supplémentaires :

- $S_D \subset S$ l'ensemble des dépôts de G
- T^{dp} la tournée du véhicule p associé au dépôt $d \in S_D$
- $\mathcal{T} = \cup_{d,p}\, T^{dp}$ l'ensemble des tournées introduites dans le problème maître réduit
- $\mathcal{B}$ l'ensemble des branchements réalisés dans la branche courante
- c^{dp} le coût de T^{dp}
- $\tilde{c}^{dp}$ le coût réduit de T^{dp}

Variables

- α^{dp} le pourcentage d'importance de la tournée T^{dp} dans la solution du problème maître réduit
- $X^{dp} = \left[x_{ij}^{dp}\right]_{(i,j)\in R} \in \{0,1\}^{|R|}$ les arcs servis ou non par T^{dp}
- $Y^{dp} = \left[y_{ij}^{dp}\right]_{(i,j)\in A} \in \mathbb{N}^{|A|}$ le nombre de fois que T^{dp} passe par un arc sans le servir
- $F^{dp} = \left[f_{ij}^{dp}\right]_{(i,j)\in A} \in \mathbb{R}_+^{|A|}$ le flot sur l'arc $(i,j) \in A$ correspondant à la demande restante dans le chemin effectué par le véhicule p associé au dépôt d

Le nombre de fois que le véhicule de la tournée T^{dp} passe sur l'arc (i,j) sera donc $x_{ij}^{dp} + y_{ij}^{dp}$.

6.2 Principe de l'algorithme (BnP-opt)

On va décomposer le GOP en d'un côté un problème maître regroupant les contraintes de service des arcs/arêtes requis ainsi que la contrainte du nombre de véhicules et d'un autre côté des problèmes esclaves. Les problèmes esclaves P_e^d correspondront à un problème où, pour un dépôt d fixé, on cherche une tournée d'un véhicule partant et arrivant au dépôt tout en respectant les contraintes de continuité de la tournée et les contraintes de capacité du véhicule. Dans le problème maître réduit P_{mr}, on prendra en compte un sous-ensemble des tournées construites dans les problèmes esclaves et on cherchera à savoir lesquelles utiliser pour minimiser le coût total. Chaque tournée T^{dp} aura un coût c^{dp} calculé de la même manière que pour le MCARP ainsi qu'un coût réduit $\tilde{c}^{dp}$ tenant compte des valeurs duales du problème maître auquel il est relié. Le calcul de ces coûts sera détaillé dans la formulation des problèmes maîtres et esclaves.

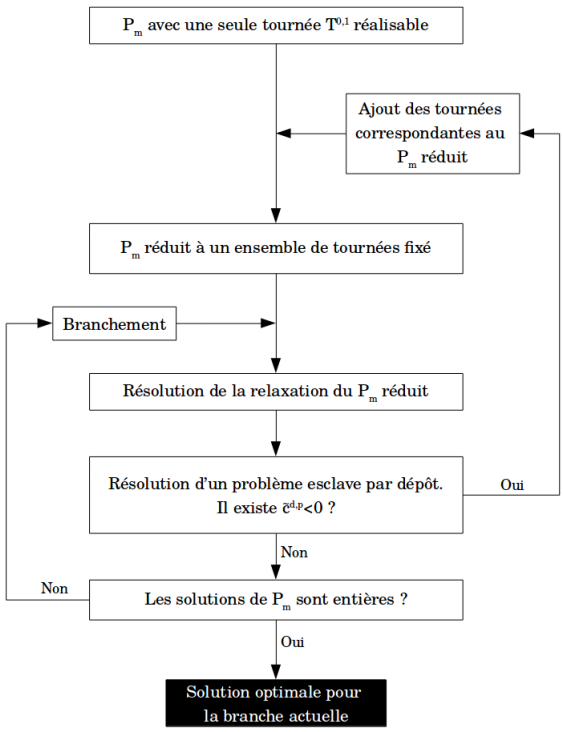


FIGURE 26 – Schéma de l'algorithme de Branch-and-Price

6.3 Formulations des problèmes maîtres et esclaves

6.3.1 Problème maître réduit (P_{mr})

On considère l'ensemble des tournées $T^{dp} \in \mathcal{T}$ introduites dans le problème maître réduit à une certaine itération de l'algorithme.

Formulation

$$\text{Minimize :} \quad \sum_{T^{dp} \in \mathcal{T}} \alpha^{dp} c^{dp} \tag{1}$$

$$\text{Subject to :} \quad \sum_{T^{dp} \in \mathcal{T}} \alpha^{dp} \leq P \tag{2}$$

$$\sum_{T^{dp} \in \mathcal{T}} \alpha^{dp} x_{ij}^{dp} \geq 1 \qquad \forall (i,j) \in A_R \tag{3}$$

$$\sum_{T^{dp} \in \mathcal{T}} \alpha^{dp} (x_{ij}^{dp} + x_{ji}^{dp}) \geq 1 \qquad \forall (i,j) \in E_R \tag{4}$$

$$\alpha^{dp} \geq 0 \qquad \forall\, T^{dp} \in \mathcal{T} \tag{5}$$

Commentaires

L'équation (1) traduit le fait qu'on veuille minimiser le coût total de toutes les tournées qu'on utilise. (2)

permet de respecter le nombre de véhicules disponibles. (3) et (4) sont les contraintes de couverture des arcs/arêtes requis. Enfin, (5) correspond aux contraintes sur les variables de choix d'utilisation d'une certaine tournée. À chaque P_{mr}, on devra ajouter les contraintes de branchements de la branche courante.

Variables duales
On récupère les différentes variables duales du problème maître afin de les utiliser dans les problèmes esclaves :

- π^p pour la contrainte sur le nombre de véhicules
- $\pi^{A_R} = \left[\pi_{ij}^{A_R}\right]_{(i,j)\in A_R}$ pour la contrainte de couverture des arcs requis
- $\pi^{E_R} = \left[\pi_{ij}^{E_R}\right]_{(i,j)\in E_R}$ pour la contrainte de couverture des arêtes requises
- $\pi^B = \left[\pi_b^B\right]_{b\in\mathcal{B}}$ pour les contraintes des branchements réalisés dans la branche courante

Le détail des branchements est expliqué par la suite.

6.3.2 Problèmes esclaves (P_e^d)

On définit un problème esclave par dépôt et pour chaque problème esclave, on cherche à trouver une tournée partant de ce dépôt qui est continue et qui respecte la capacité du véhicule. Ici, p et d seront donc fixés pour chaque (P_e^d). Le coût d'une tournée sera :

$$c^{dp} = \sum_{(i,j)\in R} x_{ij}^{dp} c_{ij} + \sum_{(i,j)\in A} y_{ij}^{dp} d_{ij}$$

On aura le coût réduit de la tournée T^{dp} associée à (P_e^d) suivant :

$$\begin{aligned}
\tilde{c}^{dp} &= c^{dp} && (1)\\
&- \pi_p && (2)\\
&- \sum_{(i,j)\in A_R} \pi_{ij}^{A_R}(i,j) x_{ij}^{dp} && (3)\\
&- \sum_{(i,j)\in E_R} \pi_{ij}^{E_R}(x_{ij}^{dp} + x_{ji}^{dp}) && (4)\\
&- \sum_{\{(i,j),d'\}\in\mathcal{B} \ \setminus \ d'=d} \pi^B_{\{(i,j),d'\}} x_{ij}^{dp} && (5)
\end{aligned}$$

Le coût réduit prend en compte le coût de la tournée auquel on soustrait des termes correspondants aux valeurs duales. (2) correspond à la contrainte du nombre de véhicules dans P_{mr}, (3) et (4) correspondent aux contraintes de service d'arcs/arêtes dans P_{mr}. Enfin, (5) correspond aux contraintes introduites lors des branchements qui portent sur le dépôt traité dans P_e^d. Les signes négatifs devant les termes des valeurs duales sont à titre indicatifs. En effet, les valeurs duales peuvent être positives ou négatives suivant le sens de l'inégalité de la contrainte correspondante dans P_{mr}. On cherchera toujours à ce que les termes des valeurs duales soient négatifs.

Formulation

$$\text{Minimize :} \quad \tilde{c}^{dp} \tag{1}$$

$$\text{Subject to :} \quad \sum_{(i,j)\in R} x_{ij}^{dp} + \sum_{(i,j)\in A} y_{ij}^{dp} = \sum_{(j,i)\in R} x_{ji}^{dp} + \sum_{(j,i)\in A} y_{ji}^{dp} \qquad \forall i \in S \tag{2}$$

$$\sum_{(0,j)\in A} y_{0j}^{dp} + \sum_{(0,j)\in R} x_{0j}^{dp} \leq 1 \tag{3}$$

$$\sum_{(j,i)\in A} f_{ji}^{dp} - \sum_{(i,j)\in A} f_{ij}^{dp} = \sum_{(j,i)\in R} x_{ji}^{dp} q_{ji} \qquad \forall i \in S \setminus d \tag{4}$$

$$\sum_{(d,j)\in A} f_{dj}^{dp} = \sum_{(i,j)\in R} x_{ij}^{dp} q_{ij} \tag{5}$$

$$\sum_{(i,d)\in A} f_{id}^{dp} = \sum_{(i,d)\in R} x_{id}^{dp} q_{id} \tag{6}$$

$$f_{ij}^{dp} \leq W(x_{ij}^{dp} + y_{ij}^{dp}) \qquad \forall (i,j) \in A \tag{7}$$

Commentaires
Dans (1), on traduit le fait qu'il faille minimiser le coût réduit de la tournée. Les contraintes (2) - (6) permettent de créer des tournées qui partent d'un dépôt, qui y reviennent, qui ne créent pas de sous-tours et qui respectent la capacité de chaque véhicule comme pour le MCARP.

6.4 Étapes importantes de l'algorithme

6.4.1 Tournée initiale

À l'initialisation de l'algorithme, on crée une tournée fictive associée à aucun dépôt, passant par tous les arcs/arêtes requis et avec un coût très élevé (10^3 fois la somme de tous les coûts de damage). Cette tournée permettra de toujours avoir une solution au problème maître. Néanmoins, si cette solution est choisie par le problème maître, c'est qu'il n'existe pas d'autres combinaisons de tournées permettant de passer par tous les arcs/arêtes requis. Le coût lié au nœud ayant pour solution cette tournée initiale ne sera jamais la solution optimale car son coût est trop élevé. Si on obtient une solution comprenant cette tournée, c'est qu'il n'existe pas de solution au GOP et qu'il faut augmenter le nombre de véhicules ou leur capacité.

6.4.2 Intégrité des solutions

Lors de la résolution du P_{mr}, la variable α^{dp} est réelle. Néanmoins, on ne peut pas avoir une tournée "non-entière". La relaxation de cette variable permet d'explorer plus vite l'arbre de branchement. Lorsque la résolution du P_{mr} conduit à des α^{dp} qui sont tous soit 0, soit 1 (les α^{dp} ne dépasseront pas 1 car avoir un $\alpha^{dp} > 1$ ne permet pas de minimiser), on aura la solution optimale pour la branche actuelle. La solution optimale du GOP sera la meilleure des solutions optimales des branches lorsqu'on ne pourra plus explorer aucun nœud.

Lorsqu'on obtient une solution non-entière au problème maître du nœud N, on trouve alors une borne inférieure $\underline{z}^N$ aux solutions dans la branche courante. Si la solution est entière, on trouve une borne supérieure globale $\bar{z}$ au problème. $\bar{z}$ sera mis à jour à chaque fois qu'on trouve des solutions entières qui améliorent le précédent $\bar{z}$. Si on trouve dans un nœud $\underline{z}^N > \bar{z}$, alors c'est qu'on ne peut pas améliorer les solutions dans cette branche et elle sera donc coupée.

6.4.3 Variables de branchement

Lors d'un branchement, nous allons créer deux nouveaux nœuds. Dans un nœud, on empêchera les tournées partant d'un dépôt de servir un arc et dans l'autre nœud, on obligera au moins un véhicule de ce dépôt de servir cet arc (en tenant compte du nombre de véhicules disponibles). Le dépôt et l'arc seront choisis à l'avance.
Dans un nœud, la contrainte introduite dans P_{mr} sera

$$\sum_p \alpha^{dp} x^{dp}_{i_B j_B} = 0 \qquad \text{avec } d \in S_D \text{ et } (i_B, j_B) \in R \text{ fixés}$$

Dans l'autre nœud, on introduira dans dans P_{mr} les contraintes

$$\begin{cases} \sum_p \alpha^{dp} x^{dp}_{i_B j_B} \geq 1 \\ \sum_p \alpha^{dp} x^{dp}_{i_B j_B} \leq P \end{cases} \quad \text{avec } d \in S_D \text{ et } (i_B, j_B) \in R \text{ fixés}$$

Pour choisir l'arc et le dépôt du branchement, on construira le vecteur $\Delta^d = \left(\sum_p \alpha^{dp} x^{dp}_{ij} \right)_{(i,j)\in R}$ pour chaque $d \in S_D$. On trouvera ensuite quelle coordonnée de quel vecteur est la plus proche de 0.5. On obtiendra alors le dépôt d et l'arc $(i_B, j_B) := (i, j)$ utilisés pour le branchement. Il faudra tout de même vérifier de ne pas ajouter deux fois le même branchement dans une branche. Un branchement $b \in \mathcal{B}$ sera donc $b = \{(i_B, j_B), d\}, \quad (i_B, j_B) \in R, \quad d \in S_D$.

Lorsqu'on introduit les contraintes $\sum_p \alpha^{dp} x^{dp}_{i_B j_B} \geq 1$, on peut être amené à rentrer en conflit avec la contrainte $\sum_p \alpha^{dp} \leq P$. Dans ce cas, le problème maître devient infaisable et la tournée initiale sera incluse dans la solution, donc l'exploration de la branche se terminera.

6.4.4 Descente dans l'arbre et complexité de résolution

À chaque nœud, on rajoute un certain nombre de tournées à celles déjà existantes qu'on transmet aux nœuds fils. Le nombre de tournées augmente donc lorsqu'on descend dans l'arbre de recherche. Comme le nombre de variables du problème maître réduit correspond au nombre de tournées du nœud, les problèmes maîtres sont donc de plus en plus longs à résoudre. De plus, plus le graphe est grand, plus l'arbre de recherche sera profond. On ne peut néanmoins pas retirer des tournées d'un nœud car celle-ci pourrait être utile dans la solution optimale. Une bonne technique d'exploration de l'arbre serait donc d'explorer quelques fois en profondeur pour fixer de bonnes bornes supérieures au départ puis d'explorer en largeur, car la résolution des problèmes maîtres prend moins de temps si on est haut dans l'arbre. On pourrait également imaginer une heuristique qui nous permettrait de retirer une tournée si elle est inactive dans les solutions de la relaxation pour un certain nombre de nœuds, quitte à devoir la réintroduire plus tard dans l'arbre. Ceci permettrait de minimiser le nombre de variables de chaque problème maître réduit. Une autre méthode d'accélération de convergence pourrait également la fixation la borne supérieure initiale en résolvant le problème dans le cas d'un unique dépôt. En effet, en ajoutant des dépôts, on ne peut pas avoir un coût plus élevé que pour un seul dépôt.

6.5 Pseudo-code

Le pseudo-code qui est présenté fait appel à de la récursivité. Dans l'implémentation du code, on utilise une pile de nœuds à explorer. Celle-ci est plus simple à gérer et permet de garder en mémoire certaines informations. Lors de l'exploration d'un nœud, on résout également la relaxation des deux nœuds fils afin de pouvoir trouver leur borne inférieure et couper directement la branche si c'est nécessaire.

Variables

- $\bar{z}$ la meilleure borne supérieure du problème
- z^N la solution de la relaxation du problème maître réduit du nœud N
- *graph* le graphe donné en entrée de l'algorithme
- P le nombre de véhicules disponibles
- $\mathcal{T}^N$ les tournées ajoutées au nœud N
- $\mathcal{B}^N$ les branchements ajoutés au nœud N
- $\mathcal{S}^N$ les solutions esclaves du nœud N
- $\mathcal{M}^N$ les solutions esclaves de $\mathcal{S}^N$ ajoutées au problème maître réduit du nœud N
- $\bar{N}$ le nœud avec la meilleure solution

Algorithm 1 Algorithme principal

Fonction BranchAndPrice(graph, P)
  initialTour, $\bar{z} \leftarrow$ CreateInitialTour(graph) ▷ Voir 6.4.1
  root $\leftarrow$ CreateRoot(initialTour)
  $z^{root} \leftarrow$ SolveRelax(root)
  Explore(root)
  Si initialTour $\notin \mathcal{T}^{\bar{N}}$ **alors** ▷ Voir 6.4.1
    La solution au problème est la solution du nœud $\bar{N}$.
  sinon
    Il n'y a pas de solution, il faut plus de véhicules ou une plus grande capacité.
  Fin Si
Fin Fonction

Algorithm 2 Exploration d'un nœud

Fonction ExploreNode(N)
  Si $z^N < \bar{z}$ **alors**
    $\mathcal{B}^N \leftarrow$ CreateBranch(N) ▷ Voir 6.4.3
    EvenNode, OddNode $\leftarrow$ CreateChildNode(N, $\mathcal{B}^N$, $\mathcal{T}^N$)
    $z^{\text{EvenNode}} \leftarrow$ SolveRelax(EvenNode)
    ExploreNode(EvenNode)
    $z^{\text{OddNode}} \leftarrow$ SolveRelax(OddNode)
    ExploreNode(OddNode)
  sinon
    La branche est coupée
  Fin Si
Fin Fonction

Algorithm 3 Résolution de la relaxation d'un nœud

Fonction SolveRelax(N)
  $\mathcal{S}^N \leftarrow \emptyset$
  Tant que HasNegativeReducedCost($\mathcal{S}^N$) ▷ Voir 6.3.1
    $\mathcal{M}^N \leftarrow$AddTours(S^N)
    z^N, DualValues $\leftarrow$ SolveMaster($\mathcal{M}^N$) ▷ Mise à jour de la borne inférieure de N
    $\mathcal{S}^N \leftarrow \emptyset$
    for Dépôt $\in S_D$ ▷ Un problème esclave résolu par dépôt
      $\mathcal{S}^N \leftarrow$ SolveSlave(Dépôt, DualValues) ▷ Voir 6.3.2
    Fin for
  Fin Tant que
  Si HasIntegerSolutions($\mathcal{M}^N$) **et** $z^N < \bar{z}$ **alors** ▷ Voir 6.4.2
    $\bar{z} \leftarrow z^N$ ▷ Mise à jour de la meilleure solution
    $\bar{N} \leftarrow N$
  Fin Si
  Retourner z^N
Fin Fonction

6.6 Algorithme dérivé et solution approchée (BnP-approx)

En plus de l'algorithme de Branch-and-Price, on propose un algorithme plus rapide permettant d'obtenir une bonne approximation de la solution. Le principe est de commencer comme l'algorithme de Branch-and-Price mais, de s'arrêter après la résolution de la relaxation de la racine. On aura donc un certain nombre de tournées ajoutées dans la racine, qui ne sont potentiellement pas retenues dans la solution de la relaxation ($\alpha^{dp} = 0$). On prendra alors l'ensemble de ces tournées et on résoudra le P_{mr} en nombre entiers ($\alpha^{dp} \in \{0, 1\} \quad \forall T^{dp} \in \mathcal{T}$) pour les tournées ajoutées dans la racine. On peut espérer qu'avec le petit nombre de tournées ajoutées, la résolution soit assez rapide. Le choix de cette méthode est motivé par le fait qu'on retrouve souvent des tournées de la solution optimale qui sont valorisés (α^{dp} proches de 1) dans la solution de la relaxation de la racine. Dès la première relaxation, on peut déjà identifier certaines bonnes tournées et on espère donc avoir une bonne solution en résolvant le PLNE directement pour la racine.

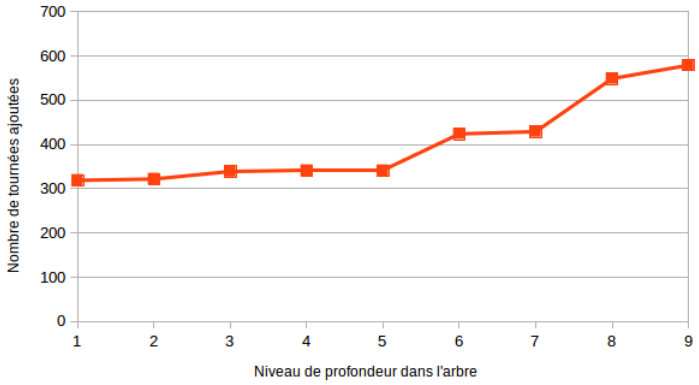


FIGURE 27 – Nombre de tournées dans les nœuds en fonction de leur profondeur dans l'abre de branchement sur un petit graphe

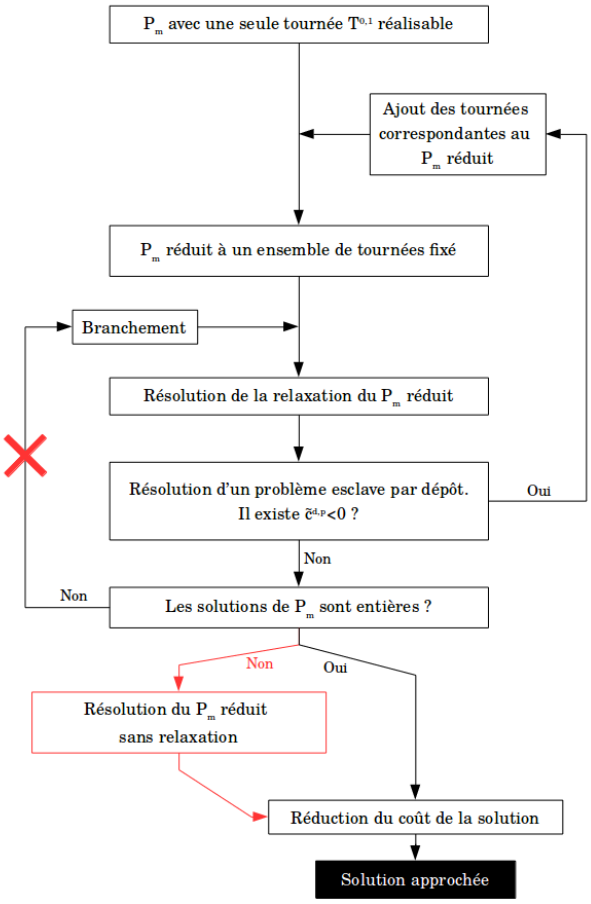


FIGURE 28 – Schéma de l'algorithme pour une solution approchée

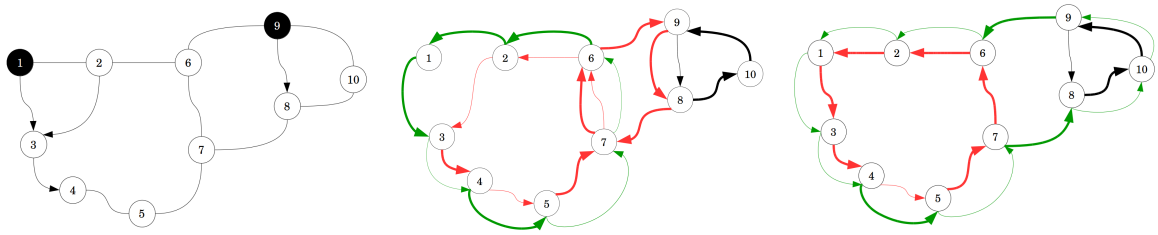


FIGURE 29 – Solution du Branch-and-Price (milieu) et solution de l'algorithme approché (droite) pour un petit jeu de données (gauche). On retrouve la tournée noire en commun. Les arcs servis sont en épais.

Comme on a une solution approchée, il se peut que plusieurs tournées servent le même arc. On rajoute donc une procédure à la fin de l'algorithme qui, si plusieurs tournées servent le même arc, assigne simplement la tournée le moins coûteuse au service de cet arc. Les autres tournées passeront simplement sur l'arc sans le servir et on pourra donc réduire un petit peu le coût global de la solution. Dans certains cas, la solution approchée requerra également plus de véhicules que la solution optimale.

7 Implémentation

Pour résoudre le problème de l'optimisation du damage dans une station de ski, les PLNE et l'algorithme de Branch-and-Price présentés précédemment ont été implémentés en `Julia 1.1` à l'aide des librairies de `JuliaOpt` (notemment `JuMP`). Ce language a l'avantage d'être assez rapide, il est développé dans le but de faire des sciences et sa structure est assez proche de `Python` ce qui rend sa prise en main assez simple. Le code est disonible à l'adresse suivante : https://github.com/TheoGuyard/Groomer-Optimization-Problem

7.1 Jeux de donnée

Les jeux de données sont stockés dans des fichiers `.txt` dont la structure est calquée sur les fichiers classique de données pour le CARP [35]. Les jeux de données commençant par `"small-"` sont des petits jeux de données adaptés d'un exemple basique [19] qui permettent de tester les programmes sur un petit graphe. Trois jeux de données ont été crées à partir de la station de ski de fond du Devoluy, de la station de Céuze en Charantes Maritimes et de la station de Gréolière dans les Alpes du sud. Les graphes liés aux jeux de données `"small-"` et des stations de ski sont donnés en annexe à la fin du rapport. Les jeux de données des stations de ski sont disponibles avec un unique dépôt ou avec plusieurs dépôts. Des jeux de données classiques du problème MCARP [35] sont aussi utilisés.

L'entête du fichier résume les informations principales de la station puis on énumère les arêtes requises, les arêtes non requises, les arcs requis et enfin les arcs non requis extraits de la modélisation sous forme de graphe du problème. Pour chaque arc/arête, on connaîtra le nœud de départ et le nœud d'arrivée, le coût de service, le coût de passage et la demande (nulle si l'arc/arête n'est pas requis). En plus de ces données, on spécifiera en entrée le nombre de dameuses disponibles. Comme on se calque sur la forme classique des jeux de données du MCARP, certaines informations de l'entête ne seront pas nécessaires pour le GOP (`DUMPING_COST`, `MAX_DURATION` et `DUMPING_SITES`) . Un exemple de jeu de données est présenté ci-dessous.

```
------------------------------------------------------------
NAME : greoliere-2
NODES : 38
REQ_EDGES : 37
NOREQ_EDGES : 23
REQ_ARCS : 3
NOREQ_ARCS : 2
CAPACITY : 20
DUMPING_COST : 0
MAX_DURATION : 0
DEPOT : 1
DUMPING_SITES : 1
LIST_REQ_EDGES :
start_node 1,end_node 2,serv_cost 5,trav_cost 1,demand 1
start_node 1,end_node 3,serv_cost 10,trav_cost 3,demand 1
start_node 3,end_node 4,serv_cost 2,trav_cost 1,demand 1
...
LIST_NOREQ_EDGES :
start_node 1,end_node 5,serv_cost 5,trav_cost 3,demand 0
...
LIST_REQ_ARCS :
start_node 13,end_node 9,serv_cost 10,trav_cost 5,demand 1
...
LIST_NOREQ_ARCS :
start_node 16,end_node 12,serv_cost 7,trav_cost 3,demand 0
...
------------------------------------------------------------
```

7.2 Structure du programme

Le programme est séparé en plusieurs modules. On entre le fichier de données et le nombre de véhicules dans `main.jl` qui transmet ces informations à `solve_problem.jl`. Avec `utils.jl`, on extraiera les données du jeu de données puis on déterminera le type du problème et l'algorithme à utiliser. Dans le cas d'un unique dépôt, on appellera `solve_single_depot.jl` qui utilisera `JuMP` pour modéliser le PLNE adéquat (via `CPP.jl`, `DCPP.jl`, `MCPP.jl` ou `MCARP.jl`). On utilisera ensuite `CPLEX` pour résoudre le PLNE. Dans le cas avec plusieurs dépôts, `solve_problem.jl` appellera `solve_multi_depot.jl` dans lequel est implémenté l'algorithme de Branch-and-Price. On pourra préciser si on veut effectuer la totalité de l'algorithme ou si on souhaite uniquement la solution approchée. Les problèmes maître et esclaves sont modélisés avec `JuMP` dans `solve_master.jl` et `solve_slave.jl`. Enfin, les chemins des véhicules seront construits à partir du nombre de passage par chaque arc/arête et les résultats seront affichés avec les fonctions implémentées dans `utils.jl`. La version `12.9` de `CPLEX` est utilisée.

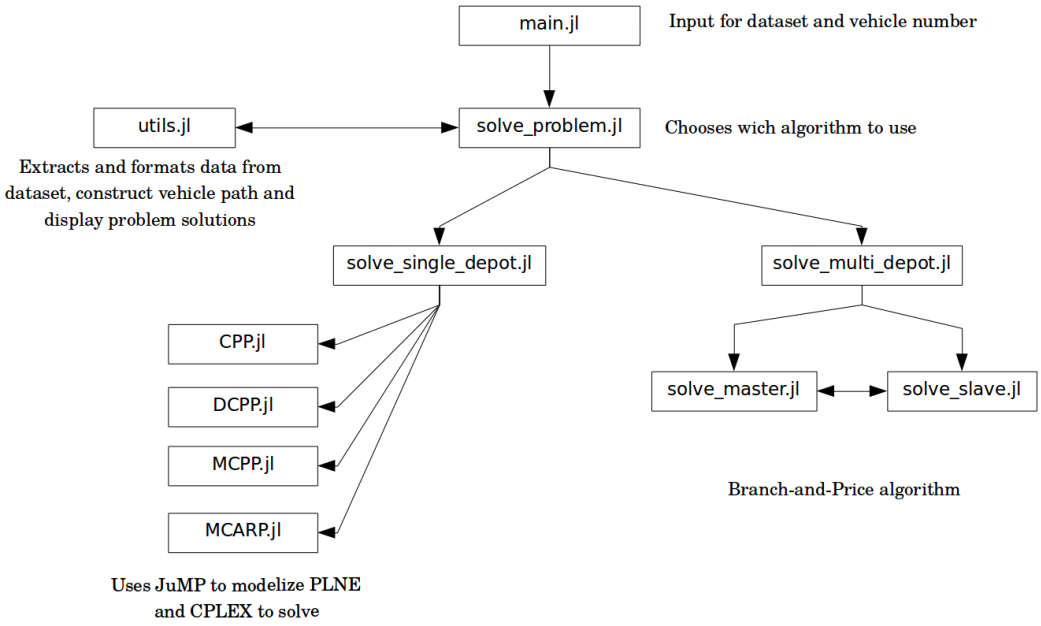
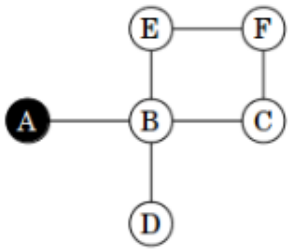


FIGURE 30 – Structure du programme

7.3 Construction des chemins pour chaque véhicule

Les résolutions de PLNE et l'algorithme de Branch-and-Price ont pour but de trouver combien de fois les dameuses doivent passer par chaque arc/arête et lesquels servir. Une fois qu'on a trouvé la solution, il faut construire le chemin par lequel doit passer chaque véhicule. Pour cela, on construit un graphe à partir et on ne laisse que les arêtes pas lequel le véhicule passe. On ajoute autant de fois l'arête que de fois où on doit y passer (on obtiendra un multigraphe dans la plupart des cas). Le graphe construit est Eulérien, on utilise l'algorithme d'Hierholzer [2] pour y trouver un cycle Eulérien. Cet algorithme n'étant pas implémenté en `Julia` pour un multigraphe, le code a été adapté de la méthode `eulerian_circuit()` implémentée dans le package `Networkx` de `Python` [30].



Véhicule	Arête	Nombre de passages
1	A-B	2
	B-C	2
	B-D	2
	A-B	2
2	B-E	1
	E-F	1
	F-C	1
	C-B	1

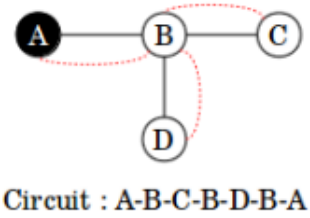


FIGURE 31 – Graphe du problème (gauche), solutions de la résolution du PLNE (milieu) et graphe construit à partir de ces solutions (droite)

8 Résultats

Les différents algorithmes sont testés sur les jeux de données. Dans le cas d'une résolution optimale pour un ou plusieurs dépôts, on notera c_{tot} le coût total de damage et t le temps de résolution en secondes. Pour les tests sur les jeux de données classiques de MCARP (`mval-...` et `Lpr-...`) [35], on notera en gras le coût total si il correspond à la valeur optimale prouvée pour le jeu de données. Dans le cas d'une résolution approchées par l'algorithme de Branch-and-Price modifié, on notera $\tilde{c}_{tot}$ le coût avant l'application de la procédure de réduction de coût et c_{tot} le coût après l'application de cette procédure. On notera également GAP l'erreur relative entre la solution approchée et la solution optimale.

Jeu de donnée	$\|S\|$	$\|E_R\|$	$\|E\|$	$\|A_R\|$	$\|A\|$	P	Algo	c_{tot}	t
small-undirected	7	10	10	0	0	1	CPP	38	0.028
small-undirected-rural	7	3	10	0	0	1	MCARP	22	0.027
small-directed	7	0	0	10	10	1	DCPP	62	0.001
small-directed-rural	7	0	0	6	10	1	MCARP	28	0.001
small-mixed	7	5	5	5	5	1	MCPP	47	0.057
small-mixed-rural	7	3	5	3	5	1	MCARP	28	0.003
devoluy-1	7	12	12	0	0	2	MCARP	60.5	0.033
devoluy-2	13	10	21	0	0	2	MCARP	67	0.068
devoluy-3	13	21	21	0	0	2	MCARP	106.5	0.111
ceuze-1	10	9	9	4	4	2	MCARP	59	0.029
ceuze-2	22	17	26	5	7	2	MCARP	122	0.051
ceuze-3	22	26	26	7	7	2	MCARP	177.5	0.199
greoliere-1	38	19	30	3	5	2	MCARP	148	0.068
greoliere-2	38	37	60	3	5	2	MCARP	239	0.218
greoliere-3	38	60	60	5	5	3	MCARP	384	15.613
mval-IF-3L-01A	24	20	20	35	35	4	MCARP	**230**	1.163
mval-IF-3L-01B	24	13	13	38	38	5	MCARP	**261**	2.583
mval-IF-3L-02A	24	16	16	28	28	4	MCARP	**324**	0.216
mval-IF-3L-02B	24	12	12	40	40	5	MCARP	**395**	6.634
mval-IF-3L-03A	24	15	15	33	33	4	MCARP	**115**	0.404
mval-IF-3L-03B	24	16	16	29	29	5	MCARP	**142**	3.641
mval-IF-3L-04A	41	26	26	69	69	5	MCARP	**580**	3.959
mval-IF-3L-05A	34	22	22	74	74	5	MCARP	**597**	13.253
mval-IF-3L-05C	34	17	17	81	81	7	MCARP	**697**	41.649
mval-IF-3L-06A	31	22	22	47	47	5	MCARP	**326**	4.234
mval-IF-3L-06B	31	22	22	44	44	6	MCARP	**317**	17.998
mval-IF-3L-07B	40	25	25	66	66	6	MCARP	**412**	3.708
mval-IF-3L-08A	30	20	20	76	76	5	MCARP	**581**	3.551
mval-IF-3L-08B	30	27	27	64	64	6	MCARP	**531**	18.248
mval-IF-3L-10A	50	32	32	106	106	5	MCARP	**634**	5.754
Lpr-IF-a-01	28	0	0	52	94	2	MCARP	**12884**	0.045
Lpr-IF-a-02	53	5	5	99	164	3	MCARP	**27152**	1.002
Lpr-IF-b-01	28	5	5	45	58	2	MCARP	**14235**	0.017
Lpr-IF-b-02	53	5	5	99	115	2	MCARP	**27754**	0.443
Lpr-IF-c-01	28	39	39	11	13	2	MCARP	**18039**	0.332

FIGURE 32 – Résolution pour un unique dépôt

Jeu de donnée	$\|S\|$	$\|S_D\|$	$\|E_R\|$	$\|E\|$	$\|A_R\|$	$\|A\|$	P	Algo	c_{tot}	t
devoluy-1	8	2	12	12	0	0	2	BnP-opt	57	12.185
devoluy-2	13	2	10	21	0	0	2	BnP-opt	61	9.558
devoluy-3	13	2	21	21	0	0	3	BnP-opt	107.5	236.006
ceuze-1	10	2	9	9	4	4	2	BnP-opt	59	20.445
ceuze-2	22	2	17	26	5	7	2	BnP-opt	114	179.202
ceuze-3	22	2	26	26	7	7	2	BnP-opt	159.5	12684.146
greoliere-1	38	2	19	30	3	5	2	BnP-opt	148	154.679

FIGURE 33 – Cas avec plusieurs dépôts résolus par Branch-and-Price de manière optimale

Jeu de donnée	$\|S\|$	$\|S_D\|$	$\|E_R\|$	$\|E\|$	$\|A_R\|$	$\|A\|$	P	Algo	c_{tot}	$\tilde{c}_{tot}$	GAP	t
devoluy-1	8	2	12	12	0	0	2	BnP-approx	60.5	60.5	5.8%	5.651
devoluy-2	13	2	10	21	0	0	3	BnP-approx	61	61	**0%**	6.088
devoluy-3	13	2	21	21	0	0	3	BnP-approx	127.5	119.5	10.0%	12.171
ceuze-1	10	2	9	9	4	4	3	BnP-approx	64	62	4.7%	5.053
ceuze-2	22	2	17	26	5	7	2	BnP-approx	116	114	**0%**	26.314
ceuze-3	22	2	26	26	7	7	3	BnP-approx	182.5	169.5	5.9%	37.210
greoliere-1	38	2	19	30	3	5	3	BnP-approx	163	162	8.6%	18.767

FIGURE 34 – Cas avec plusieurs dépôts résolus par Branch-and-Price de manière approchée

8.1 Impact du nombre de véhicules et de dépôts

Les comportements des méthodes de résolution (résolution par solver ou Branch-and-Price) sont très différentes vis-à-vis de la variation des données en entrée. En effet, lorsqu'on résout avec le solver, on utilise des variables et des contraintes indexées par le nombre de véhicules. Rajouter un véhicule signifie rajouter des variables et des contraintes donc le solver mettra plus de temps à résoudre le problème. C'est pourquoi dans les instances `mval-...` et `Lpr-...`, il a été impossible de résoudre dans un temps raisonnable les problèmes quand le nombre de véhicule devenait trop grand.

Dans la résolution par Branch-and-Price, ajouter un véhicule n'a quasiment aucun effet sur le temps de résolution. En effet, le nombre de problèmes esclaves résolus dépend uniquement du nombre de dépôt. En rajoutant des véhicules, on modifiera simplement une contrainte et on résoudra un problème moins contraint ce qui peut amener à une résolution plus rapide. Rajouter un dépôt signifie résoudre un problème esclave en plus à chaque itération. Comme ces problèmes sont assez rapides à résoudre, le temps de résolution augmente très peu lorsqu'on rajoute des dépôts.

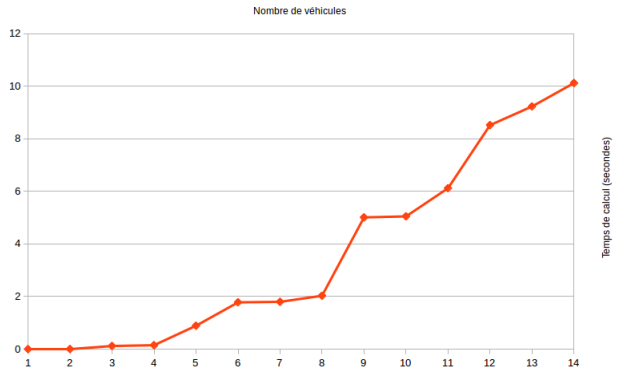


FIGURE 35 – Impact du nombre de véhicules pour ceuze-2 (dépôt unique, résolution par solver)

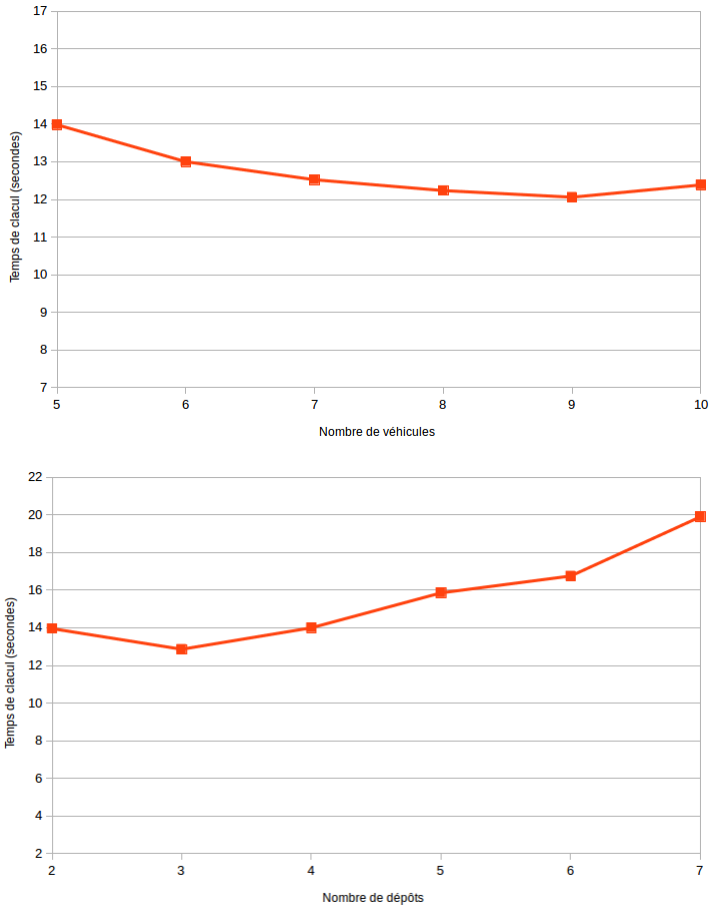


FIGURE 36 – Impact du nombre de véhicules (avec 2 dépôts, résolution par Branch-and-Price) et impact du nombre de dépôts (avec 3 véhicules, résolution par Branch-and-Price) pour devoluy-3

8.2 Comparaison des méthodes de résolution

La résolution par solver est nettement plus rapide que le Branch-and-Price mais ne permet pas que de traiter les cas avec plusieurs dépôts. De plus l'algorithme de Branch-and-Price n'a pas été implémenté de manière à optimiser le temps de résolution mais plutôt de telle sorte à comprendre ce qui se passe à chaque étape. On peut donc espérer améliorer son temps de résolution en se penchant plus sur le code. L'étude de l'influence des paramètres su l'algorithme de Branch-and-Price est très intéressant. En effet, si une nouvelle implémentation du Brach-and-Price permet de réduire le temps de résolution, on aura un algorithme qui sera rapide et peu sensible aux paramètres de départ. On pourra donc ajouter des dépôts et des véhicules sans trop impacter le temps de résolution. Ce n'est pas le cas pour la résolution par solver. On peut également noter que l'algorithme de Branch-and-Price modifié donne d'assez bonnes solutions en un temps raisonnable. L'erreur commise est en moyenne de 5%. Dans certains cas, on obtient une solution avec le même coût mais qui nécessite plus de dameuses (la solution est donc moins bonne).

9 Conclusion

Lors de ce stage de 8 semaines, j'ai été amené à résoudre un problème de damage dans une stations de ski. Les premières semaines ont été quasiment exclusivement consacrées aux recherches bibliographiques. Celle-ci ont également continuées tout au long du stage. Grâce à cela, j'ai pu trouver une modélisation adéquat au problème et justifiée par les différentes contraintes pratiques qui étaient imposées. J'ai aussi pu identifier des problèmes déjà existants qui ont aiguillé la modélisation de telle sorte à se rapprocher de ces problèmes. J'ai ensuite été amené à comprendre, mettre en place et implémenter plusieurs méthodes de résolution à base de programmes linéaires pour résoudre les différents types de problèmes liés au GOP. Pour cela, j'ai dû me renseigner sur les méthodes déjà existantes et les comprendre. Devoir coder l'algorithme de Branch-and-Price sans aide de packages extérieurs m'a aussi permis de bien comprendre chaque étape de l'algorithme ainsi que son fonctionnement. Cela a également été l'occasion d'apprendre un nouveau langage de programmation dédié principalement aux mathématiques. Le GOP m'a permis de découvrir un assez large panel de ce qui existe en terme de problème de routage, les méthodes de résolution développées et m'a également permis de me familiariser avec certains outils utilisés dans la recherche ou dans l'industrie comme par exemple le solver `CPLEX`. Le fait de travailler sur un problème concret et de lire des articles de recherche très récents a grandement participé à la conservation de ma motivation tout au long du stage. Ces deux mois de stage sont donc, pour moi, une très bonne première approche du monde de la recherche appliquée en mathématique.

10 Annexes

10.1 Jeux de données small

Sur les graphes suivantes, le nœud correspondant au dépôt est le cercle noir. Les pistes à damer sont représentées en rouge et les autres pistes en noir. Les pistes praticables dans les deux sens sont représentées par des traits et celles praticables dans un seul sens par des flèches.

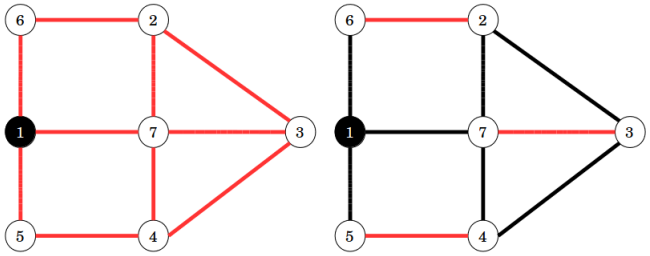


FIGURE 37 – small-undirected et small-undirected-rural

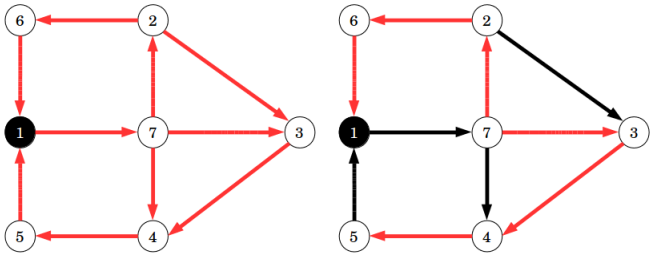


FIGURE 38 – small-directed et small-directed-rural

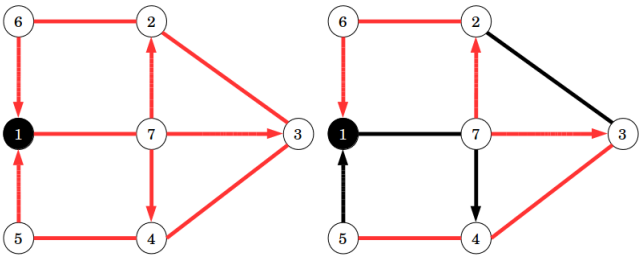


FIGURE 39 – small-mixed et small-mixed-rural

10.2 Modélisation des stations de ski

Pour les trois stations de ski étudiées, les pistes sont représentées avec la couleur qui correspond à leur difficulté afin de mieux se repérer. Les pistes requises ne sont pas repérées d'une manière particulière car ce ne sont pas les mêmes pour les trois jeux de données pour chacune des stations. Les pistes sont représentées par une flèche si elles sont praticables dans un seul sens et par un trait si elles le sont dans les deux sens. Le choix des pistes praticables dans un seul sens est arbitraire, il a été décidé que toutes les pistes noires (qui sont donc pentues) et certaines pistes rouges sont praticables uniquement en descendant à l'exception près de la station de Dévoluy qui est une station de ski de fond (les pentes ne sont donc pas importantes).

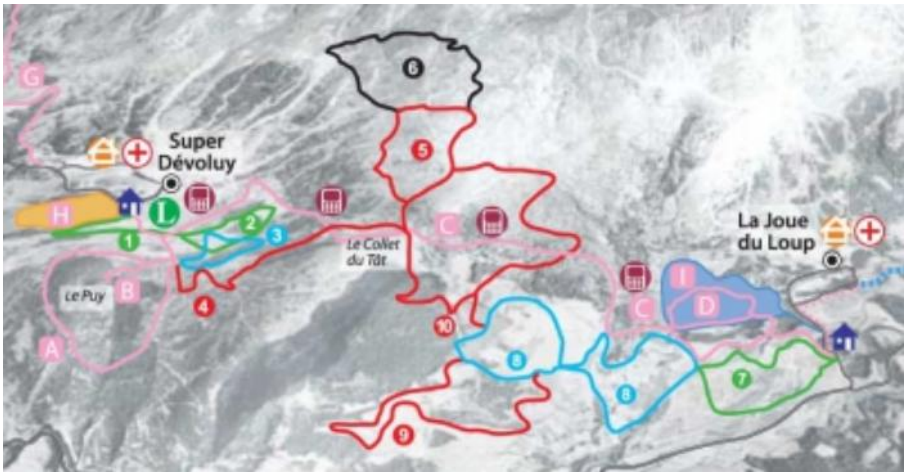


FIGURE 40 – Station de ski du fond du Dévoluy

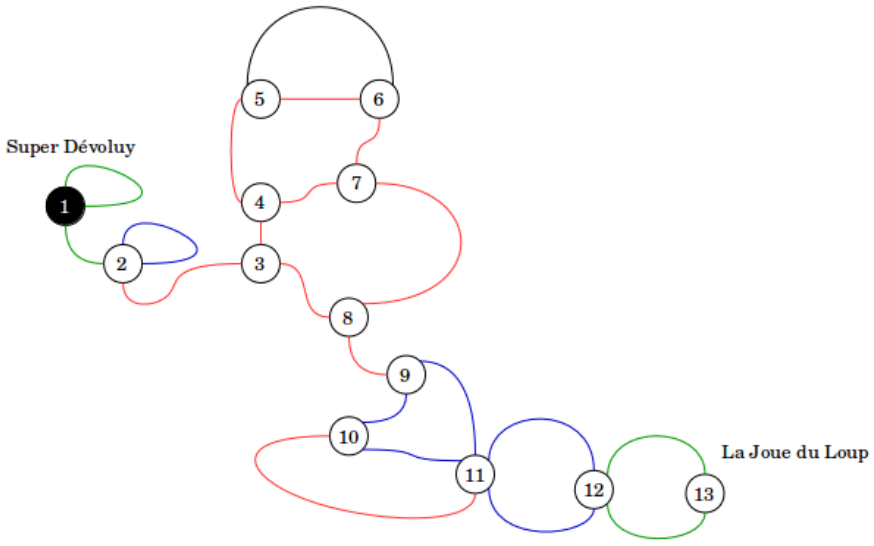


FIGURE 41 – Station de ski du fond du Dévoluy sous forme de graphe

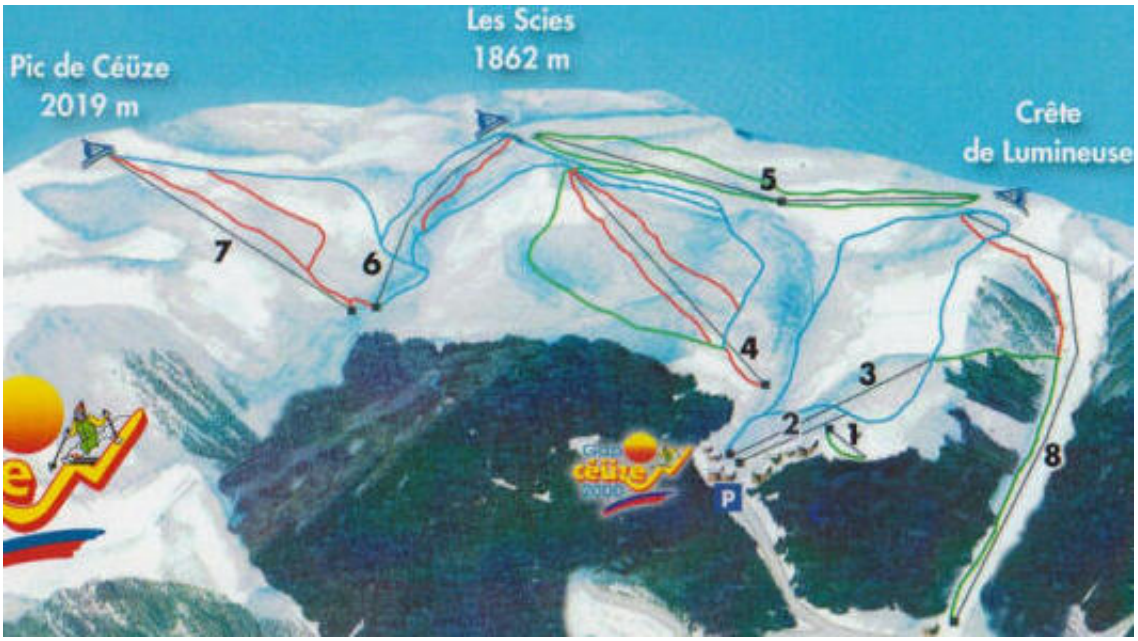


FIGURE 42 – Station de ski de Céuze

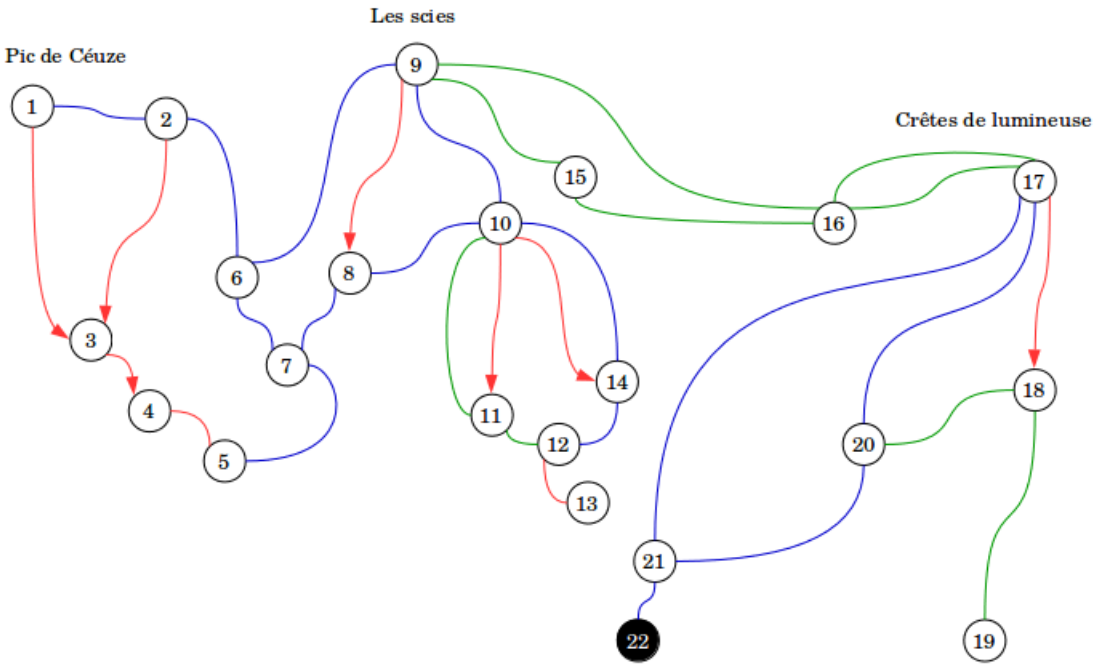


FIGURE 43 – Station de ski de Céuze sous forme de graphe

FIGURE 44 – Station de ski de Gréolière

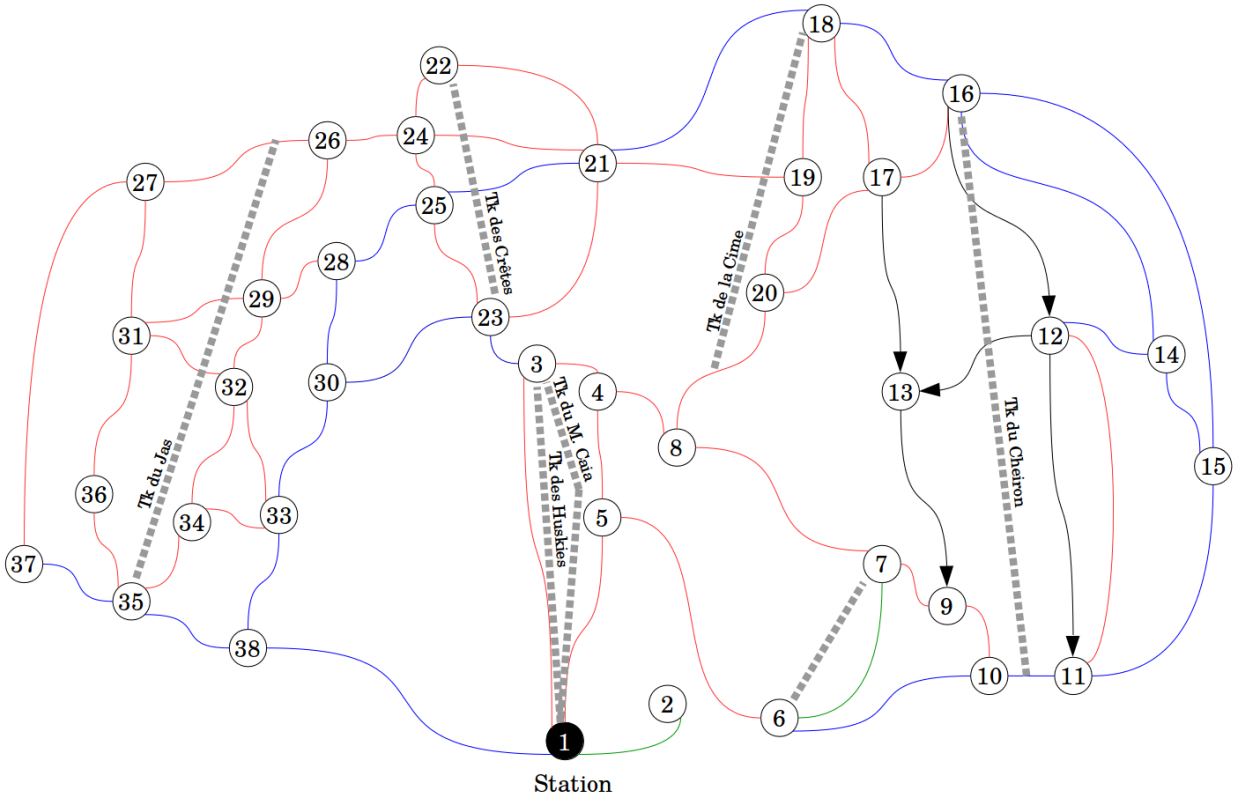


FIGURE 45 – Station de ski de Gréolière sous forme de graphe

Références

[1] M. Mittaz A. Hertz, G. Laporte. A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48 :129–135, 2000.

[2] Anonymous. Eulerian path. https://en.wikipedia.org/wiki/Eulerian_path#Hierholzer's_algorithm, 2018.

[3] José M. Belenguer and Enrique Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30(5) :705 – 728, 2003.

[4] Nicos Christofides, Vicente Campos, Angel Corberán, and E Mota. An algorithm for the rural postman problem. 1981.

[5] M.J. Todd D. Goldfarb. Linear programming. *Handbooks in ope- rations research and management science*, 1 :73–170, 1989.

[6] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 1965.

[7] Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1) :88–124, Dec 1973.

[8] L. Euler. *Solutio problematis ad geometriam situs pertinentis*. 1759.

[9] Les Foulds, Humberto Longo, and Jean Martins. A compact transformation of arc routing problems into node routing problems. *Annals of Operations Research*, 226(1) :177–200, Mar 2015.

[10] D.R. Fulkerson. An out-of-kilter method for solving minimal cost flow problems. *Journal of the Society for Industrial and Applied Mathemetics*, 1961.

[11] Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3) :305–315.

[12] R.E. Gomory. Outline of an algorithm for integer solutions to linear. *Bulletin of the American Mathematical society*, 64 :275–278.

[13] N. Karmarkar. A new polynomial time algorithm for linear pro- gramming. *Combinatorica*, 4 :373–395.

[14] Byung-In Kim, Seongbae Kim, and Surya Sahoo. Waste collection vehicle routing problem with time windows. *Computers and Operations Research*, 33(12) :3624 – 3642, 2006. Part Special Issue : Recent Algorithmic Advances for Arc Routing Problems.

[15] Vladimir Kolmogorov. Blossom v : A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 2010.

[16] Dmitry Krushinsky and Tom Van Woensel. An approach to the asymmetric multi-depot capacitated arc routing problem. *European Journal of Operational Research*, 244(1) :100 – 109, 2015.

[17] L. Santiago Pinto L. Gouveia, M. Candida Mourao. Lower bounds for the mixed capacited arc routing problem. *Computers and Operations Research*, pages 692–699.

[18] Philippe Lacomme, Christian Prins, and Alain Tanguy. First competitive ant colony scheme for the carp. In Marco Dorigo, Mauro Birattari, Christian Blum, Luca Maria Gambardella, Francesco Mondada, and Thomas Stützle, editors, *Ant Colony Optimization and Swarm Intelligence*, pages 426–427, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[19] M. Minoux M. Gondran. *Graphes et algorithmes*. Études et recherches d'Électricité de france edition.

[20] M. Minoux M. Gondran. *Graphes et algorithmes*, chapter 8, page 328. 1995.

[21] M. Minoux M. Gondran. *Graphes et algorithmes*, chapter 8, page 312. 1995.

[22] M. Minoux M. Gondran. *Graphes et algorithmes*, chapter 8, page 314. 1995.

[23] M. Minoux M. Gondran. *Graphes et algorithmes*, chapter 7, page 292. 1995.

[24] Rafael Martinelli, Diego Pecin, Marcus Poggi, and Humberto Longo. A branch-cut-and-price algorithm for the capacitated arc routing problem. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, pages 315–326, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[25] Kwan Mei-Ko. Graphic programming using odd or even points. *Acta Mathematica Sinica (in Chinese)*, (10) :263–266.

[26] M. Minoux. *Programmation mathématique*, volume 2, chapter 8, page 59. Cnet edition.

[27] M. Minoux. *Programmation mathématique*, volume 2, chapter 8, page 73. Cnet edition.

[28] M. Minoux. *Programmation mathématique, théorie et algorithmes*.

[29] M. Cândida Mourão and Leonor S. Pinto. An updated annotated bibliography on arc routing problems. *Networks*, 70(3) :144–194.

[30] M. Trenfield N. Mohammadi, A. Hagberg. Networkx eulerian circuit method. https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/euler.html, 2010.

[31] Wen Lea Pearn. Solvable cases of the k-person chinese postman problem. *Operations Research Letters*, 16(4) :241 – 244, 1994.

[32] W.L. Pearn and T.C. Wu. Algorithms for the rural postman problem. *Computers and Operations Research*, 22(8) :819 – 828, 1995.

[33] G. J. Minty V. Klee. How good is the simplex algorithm. *Academic Press, New-York*, page 159–175.

[34] L.J. White. Minimum cover of fixed cardinality in weighted graphs. 1971.

[35] Elias J. Willemse and Johan W. Joubert. Benchmark dataset for undirected and mixed capacitated arc routing problems under time restrictions with intermediate facilities. *Data in Brief*, 8 :972 – 977, 2016.

[36] Sanne Wøhlk. *A Decade of Capacitated Arc Routing*, pages 29–48. Springer US, Boston, MA, 2008.