Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut Supérieur de l'Aéronautique et de l'Espace

---

Présentée et soutenue par

François LAMOTHE

Le 29 novembre 2021

Le problème de flot insécable: application à la gestion des communications d'une constellation de satellites.

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Mathématiques et Applications**

Unité de recherche :
ISAE-ONERA MOIS MOdélisation et Ingénierie des Systèmes

Thèse dirigée par
Alain HAIT et Emmanuel RACHELSON

Jury

M. Marc SEVAUX, Rapporteur
Mme Safia KEDAD-SIDHOUM, Rapporteure
M. Cédric BAUDOIN, Examinateur
M. Alain HAIT, Directeur de thèse
M. Emmanuel RACHELSON, Co-directeur de thèse
M. Viet Hung NGUYEN, Président

Déclarations

Ce travail a été financé par le CNES et Thales Alenia Space. De plus, il fut réalisé en collaboration avec plusieurs de leurs membres.

Remerciements

Je remercie mes directeurs de thèse Alain Haït et Emmanuel Rachelson pour avoir été présents du début à la fin, pour avoir été de très bon guides, pour avoir été patients avec mes défauts et surtout pour tout ce qu'ils m'ont appris.

Je remercie mes collègues de SUPAERO parce que c'est grâce à eux que ces trois dernières années se sont aussi bien passée et que je les garderais en mémoires parmi les meilleures périodes de ma vie.

Je remercie ma famille pour leur support, pour m'avoir donner les conseils dont j'avais besoin et car je lui dois une part essentielle de qui je suis maintenant.

Je remercie Cédric Baudoin, Mathieu Gineste, Thibault Deleu et Jean-Baptiste Dupé pour m'avoir accompagné sur ce chemin. Ce fut un vrai plaisir de travailler avec vous et de vous voir à chaque fois.

Je remercie Bernard Gendron et Claudio Contardo pour leur temps, leurs discussions, leurs idées et leurs conseils.

Enfin, je remercie le TéSA de donner un aussi bon cadre à des thèses comme la mienne et je remercie ses membres de former une communauté aussi dynamique et accueillante.

Table des matières

Notations

Nous présentons ici l'ensemble des notations utilisées dans ce manuscrit à partir du Chapitre 2. En effet, dans le Chapitre 1 dédié à l'application industrielle, nous utilisons des notations différentes qui sont précisées sur le moment. Dans le reste de la thèse nous utiliserons les notations suivantes.

Lorsque nous discuterons de flots insécables statiques et donc de graphes statiques, nous noterons $G = (V, E)$ un graphe orienté ou non, avec V son ensemble de nœuds et E son ensemble d'arcs. Lorsque nous discuterons de flots insécables dynamiques et donc de graphes dynamiques, nous noterons $G = (V, (E_t)_{t \in T})$ un graphe dynamique orienté ou non orienté où V est son ensemble de nœuds et $(E_t)_{t \in T}$ son ensemble d'arcs autorisés à chaque pas de temps.

Nous utiliserons les indices suivants :

- $t \in T$ pour le temps ;
- $k \in K$ pour les commodités ;
- $v \in V$ pour les sommets des graphes ;
- $e \in E$ ou $e \in E_t$ pour les arcs des graphes ;
- $p \in P^k$ ou $p \in P^k_t$ pour les chemins de chaque commodité ;
- $s \in S^k$ pour les séquences de chemins de chaque commodité ;
- $g \in G = \{0, 1\}^K$ pour les patrons de commodités.

Nous utiliserons les constantes suivantes :

- $(O^k, D^k, d^k)_{k \in K}$ est un ensemble de commodités définies par leur origine, leur destination et leur demande ;
- $((O^k_t, D^k_t, d^k_t)_{t \in T})_{k \in K}$ est un ensemble de commodités définies par leur origine, leur destination et leur demande à chaque pas de temps ;
- $(c_e)_{e \in E}$ sont les capacités des arcs d'un graphe statique ;
- $(c_{et})_{e \in E_t, t \in T}$ sont les capacités des arcs d'un graphe dynamique ;
- $c_{min} = \min_{e \in E} c_e$ est la plus petite capacité des arcs d'un graphe statique ;
- $d_{max} = \max_{k \in K} d^k$ est la plus grande demande des commodités ;
- o^g_e est le dépassement de capacité sur l'arc e induit par un patron de commodités g ;
- B est la quantité de dépassement autorisée a être induit par l'ensemble des commodités sur un pas de temps sans encourir de pénalisation dans les programmes linéaires (en nombres entiers) ;
- n^k_s est le nombre de changements de chemin induit par une séquence de chemins s de la commodité k ;
- p^k est le dernier chemin utilisé par la commodité k avant l'horizon d'un problème de flot insécable dynamique ;

— C^* représente la valeur optimale d'un problème de flot insécable ;

— Δ^* représente la valeur optimale de la relaxation linéaire d'un problème de flot insécable ;

— $\gamma = \frac{d_{max}}{c_{min}\Delta^*}$ est un paramètre de granularité de la demande des commodités par rapport à la capacité des arcs ;

— δ_x^y est la notation de Kronecker, égale à 1 si $x = y$ et à 0 sinon.

Nous utiliserons les ensembles suivants :

— $E^-(v)$ est l'ensemble des arcs entrants du nœud v ;

— $E^+(v)$ est l'ensemble des arcs sortants du nœud v ;

— P^k ou P_t^k est l'ensemble des chemins valides pour la commodité k ;

— $S^k = \Pi_{t \in T} P_t^k$ est l'ensemble des séquences de chemins valides pour la commodité k ;

— S_{et}^k est l'ensemble des séquences de chemins valides pour la commodité k utilisant l'arc e au temps t ;

Nous utiliserons les variables suivantes dans les programmes linéaires en nombres entiers :

— f_e^k indique si la commodité k utilise l'arc e pour transmettre son flot ;

— x_p^k ou x_{pt}^k indique si la commodité k utilise le chemin p pour transmettre son flot ;

— y_e^g indique si le patron de commodités g est autorisé sur l'arc e ;

— x_s^k indique si la commodité k utilise la séquence de chemins s pour transmettre son flot dans le temps ;

— n_{pt}^k indique si la commodité k utilise le chemin p au pas de temps t mais pas au pas de temps $t-1$;

— o_e ou o_{et} indique le dépassement de capacité sur un arc e ;

— o_t représente le montant du dépassement qui excède, au pas de temps t, le montant B de dépassement non pénalisé.

De plus, dans le Chapitre 5 nous considérerons les notations suivantes :

— x est le vecteur des variables d'un programme linéaire en nombres entiers ;

— A_1 et A_2 sont les matrices des contraintes d'un programme linéaire en nombres entiers ;

— b_1 et b_2 sont les seconds membres des contraintes d'un programme linéaire en nombres entiers ;

— X est un produit d'ensembles $\mathbb{R}$ et $\mathbb{Z}$;

— $\bar{X}$ est la relaxation de l'ensemble X où les ensembles $\mathbb{Z}$ sont remplacés par des ensembles $\mathbb{R}$;

— $LR_1 = \{x \in \bar{X} | A_1 x \leq b_1\}$ est le polyèdre induit par la matrice A_1 et le second membre b_1 ;

— $LR_2 = \{x \in \bar{X} | A_2 x \leq b_2\}$ est le polyèdre induit par la matrice A_2 et le second membre b_2 ;

— $Q_1 = conv(\{x \in X | A_1 x \leq b_1\})$ est l'enveloppe convexe des points entiers contenus dans LR_1 ;

— $Q_2 = conv(\{x \in X | A_2 x \leq b_2\})$ est l'enveloppe convexe des points entiers contenus dans LR_2 ;

— $\mathcal{C}(Q_2)$ est l'ensemble des coupes valides pour le polyèdre Q_2 ;

— λ_i est la variable représentant le coefficient d'un point x_i dans une combinaison convexe ; cette variable est celle d'un programme linéaire (en nombres entiers) utilisant une formulation de type Dantzig-Wolfe ;

— π est la variable duale associée à la contrainte $\sum_{i \in I} \lambda_i x_i = x$ dans une formulation de Dantzig-Wolfe ; c'est aussi le vecteur des coefficients des coupes de Fenchel car ces deux objets sont fortement reliés ;

— π_0 est la variable duale associée à la contrainte $\sum_{i \in I} \lambda_i = 1$ dans une formulation de Dantzig-Wolfe ; c'est aussi le second membre des coupes de Fenchel car ces deux objets sont fortement reliés ;

— $\hat{x}$ un point devant être séparé du polyèdre Q_2 ;

— $\bar{x}$ un point appartenant au polyèdre Q_2 utilisé comme paramètre d'une normalisation directionnelle ;

— $Q_2^f(\hat{x}) = \{x \in Q_2 | x_i = \hat{x}_i \text{ si } \hat{x}_i \in \{0, 1\}\}$ est le sous-polyèdre de Q_2 où toutes les variables prenant une valeur binaire dans $\hat{x}$ sont fixées à leur valeur dans $\hat{x}$;

— $\angle(x, y)$ est l'angle entre deux vecteurs x et y.

Introduction

Le problème de la transmission de ressources indivisibles au travers d'un réseau est un problème générique présent dans de nombreuses applications. En effet, ce type de problème se retrouve dans des industries telles que le transport de fret ou encore les télécommunications (réseaux optiques, communications satellitaires ...). L'amélioration des méthodes de résolution pour ce problème représente donc un enjeu important qu'il convient d'aborder sous plusieurs angles. Premièrement, améliorer la qualité des solutions trouvées permet d'augmenter l'efficacité des systèmes que ce problème affecte. Deuxièmement, accélérer la résolution de ce problème est important dans les applications où le temps de calcul est très restreint, mais aussi lorsque les instances du problème de transmission considéré sont de grande taille.

Ce dernier point représente un enjeu important dans l'application industrielle qui motive cette thèse : la constellation de satellites de télécommunication Telesat. En effet, cette industrie tend à construire des constellations contenant de plus en plus de satellites afin d'augmenter le débit internet que le système est capable de transmettre. On peut ainsi constater cette évolution en comparant les 66 satellites de la constellation Iridium (2018) avec les 288 satellites de la constellation Telesat (à venir en 2022) ou encore les 10 000 satellites envisagés dans la constellation Starlink (annoncée pour 2024). En parallèle de l'augmentation du nombre de satellites, on constate aussi une augmentation du nombre d'utilisateurs de ces constellations. Celle ci s'explique à la fois par l'accroissement de la richesse de la population, l'essor de nouvelles applications telles que les accès internet dans les avions ou les bateaux mais aussi tout simplement par l'augmentation de la capacité et de la qualité des services de télécommunication par satellite. La combinaison de ces facteurs tend à créer des problèmes de transmission de ressources de plus en plus difficiles à résoudre ce qui nécessite des algorithmes de résolution plus performants.

Or, dans le cadre de la constellation Telesat que nous étudions en partenariat avec Thalès Alenia Space et le Centre National d'études spatiales, le débit total transmis par la constellation est estimé aux alentours de 7 Térabits par seconde. Si l'on considère qu'un utilisateur moyen demande aux alentours de 5 Mégabits par seconde, une hausse de 5% de la capacité de transmission de la constellation due à une meilleure gestion des ressources de communications représente une possibilité d'accès au service de la constellation pour des centaines de milliers d'utilisateurs supplémentaires. Ce chiffre peut sembler faible en comparaison avec la population mondiale actuelle, mais une telle constellation n'est pas destinée à concurrencer le réseau Internet terrestre chargé de fournir la majeure partie du débit demandé. En effet, le rôle de la constellation Telesat est de compléter le réseau terrestre dans les zones où celui-ci est trop cher à construire ou pour les utilisateurs inaccessibles tels que ceux au milieu des océans.

Dans cette thèse, nous nous intéressons au problème de transmission de la ressource indivisible qu'est le débit des utilisateurs dans une constellation. Ce problème correspond à un problème classiquement étudié dans la littérature des problèmes de flots, sous le nom de

problème de flot insécable. Bien que ses propriétés théoriques soient bien connue et que de nombreuses approches de résolution existent, celles-ci manquent d'efficacité lorsque la taille du problème est importante. Nous tentons de combler cette lacune en proposant des algorithmes présentant de bonnes performances sur de grandes instances de ce problème. D'autre part, l'introduction de la dynamique de la constellation dans le problème nous mène à nous intéresser au problème de flot insécable dynamique. Ce problème est peu étudié dans la littérature, c'est pourquoi nous étendons l'ensemble des méthodes de résolution testées sur ce problème en proposant différentes approches et en les comparant expérimentalement sur des jeux d'instances que nous proposons. Enfin, nous étudions des méthodes de décomposition permettant de renforcer la relaxation linéaire du problème flot insécable. En effet, cette relaxation linéaire est à la base de la plupart des méthodes de résolution proposées dans la littérature. Le calcul d'une relaxation puissante est donc un enjeu de la résolution du problème de flot insécable. Après avoir présenté et réimplémenté deux méthodes de la littérature, nous proposons une nouvelle méthode de décomposition s'inspirant des deux méthodes précédentes. Une étude empirique montre que la nouvelle méthode proposée possède un avantage compétitif important sur les grandes instances du problème de flot insécable.

Ce manuscrit est divisé en cinq chapitres. Dans le premier chapitre, nous présentons en détail le problème de gestion des ressources de télécommunication dans une constellation de satellites et montrons que le problème est trop grand pour être résolu directement. En restreignant notre étude à une sous-partie de ce problème, nous montrons qu'il convient d'étudier les problèmes de flot insécable statique et dynamique. Le deuxième chapitre de ce manuscrit est consacré à une revue de la littérature. Nous rappelons les formulations envisagées pour les problèmes de flot insécable statique et dynamique ainsi que les divers algorithmes proposés pour les résoudre. De plus, nous présentons une méthode centrale à la résolution de ce type de problème : la génération de colonnes. Cette présentation sera faite dans le cadre des programmes linéaires généraux avant d'être appliquée dans le cas particulier des flots insécables statique et dynamique. Dans le troisième chapitre, nous présentons la première contribution de cette thèse. Il s'agit d'une heuristique pour le problème de flot insécable basée sur l'arrondi aléatoire (*randomized rounding*). Une variante de cette heuristique est aussi étudiée afin de montrer qu'elle possède des garanties d'approximation égalant les meilleures garanties d'approximations de la littérature. Enfin, ces algorithmes sont comparés empiriquement aux propositions de la littérature sur des instances de grande taille. Cette contribution a donné lieu à un article ayant été accépté dans le *Journal of heuristics* (Lamothe *et al.*, 2021). Le quatrième chapitre est consacré à la présentation de notre contribution pour le problème de flot insécable dynamique. Nous proposons des formulations pour résoudre ce problème et sa relaxation linéaire, complétant ainsi la formulation proposée dans la littérature. De plus, nous étudions de nouveaux algorithmes pour résoudre le problème de *pricing* de la formulation proposée dans la littérature. Enfin, nous introduisons plusieurs solveurs basés sur ces formulations et les comparons empiriquement en mettant en valeur certains de leurs aspects clés. Cette étude donnera lieu à une publication sous la forme d'un article de journal qui est pour le moment en préparation (Lamothe *et al.*). Le cinquième et dernier chapitre de ce manuscrit étudie des méthodes de décomposition applicables au problème de flot insécable dans le but d'en améliorer la relaxation linéaire. Nous commençons par passer en revue deux méthodes de la littérature : la décomposition de Dantzig-Wolfe et la décomposition de Fenchel. Nous

proposons ensuite une nouvelle méthode de décomposition utilisant des idées venant des deux méthodes précédentes. L'ensemble de ces méthodes est ensuite comparé empiriquement sur des instances du problème de flot insécable.

Chapitre 1

Description du problème de la constellation de satellites

Résumé du chapitre

Dans ce chapitre, nous faisons une description détaillée du cas d'étude ayant mené aux travaux de cette thèse : la constellation de satellites de télécommunications Telesat. Après avoir présenté généralement la constellation, nous détaillons trois axes principaux du problème de gestion des télécommunications de cette constellation : (i) la gestion des liens satellite-utilisateur, (ii) la gestion des liens intersatellitaires et satellite-sol, (iii) et l'impact de la dynamique de la constellation sur les axes précédents. Puis, nous proposons une modélisation de programmation linéaire en nombres entiers pour le problème complet cependant, au vu de la taille du problème, celui-ci ne peut pas être abordé frontalement. C'est pourquoi nous concentrons dans le reste de ce manuscrit sur les deux premiers axes du problème ce qui nous conduit à nous intéresser aux problèmes de flot insécable statique et dynamique.

La motivation ayant mené aux travaux de cette thèse sur les flots insécables provient d'un cas d'étude sur la constellation de satellites de télécommunications Telesat. Cette constellation se compose d'un ensemble de petits satellites orbitant la Terre à basse altitude et communiquant entre eux. Ce système a pour but de servir d'alternative au réseau terrestre d'Internet. En effet, dans les régions peu peuplées ou lorsque le coût de construction est très élevé, le déploiement d'un réseau Internet terrestre peut ne pas s'avérer pertinent. En comparaison, une constellation est adaptée à servir des utilisateurs faiblement concentrés spatialement et répartis dans le monde entier. Sa capacité à fournir du débit dépend peu des coordonnées géographiques et des conditions au sol. Une constellation est donc parfaitement adaptée pour compenser les faiblesses d'un réseau terrestre. Un autre type d'application envisagé est de fournir du débit à des objets en mouvement, en particulier dans les océans. Ainsi, il serait possible de fournir du débit à des bateaux ou à des avions offrant ainsi un nouveau service à bord de ces moyens de transport.

Pour mener à bien sa tâche, la constellation Telesat est composée de 288 satellites et de 100 stations sol qui servent de point de connexion au réseau terrestre. De plus, chaque utilisateur est équipé d'une antenne qui sert à communiquer avec les satellites. Tous ces éléments

sont interconnectés en un réseau de communication illustré en Figure 1.1. Les satellites sont répartis sur deux constellations distinctes ayant chacune une topologie en grille torique ce qui est illustré en Figure 1.2. La première, contenant les deux tiers des satellites, couvre une grande région autour de l'équateur et s'appelle la constellation inclinée. Les satellites restants composent la constellation polaire. Ils orbitent sur plusieurs méridiens et viennent compenser les faiblesses de la constellation principale au niveau des pôles. Avec une altitude avoisinant les 1000 km pour la constellation polaire et 1250 km pour la constellation inclinée, la constellation Telesat appartient à la catégorie des constellations *Low Earth Orbit* (LEO, 1000-1400 km). Cette catégorie est intermédiaire entre les constellations *Medium Earth Orbit* (MEO, 8000 km) et *Very Low Earth Orbit* (VLEO, 350 km) qui composent le reste des constellations lancées et envisagées à ce jour.

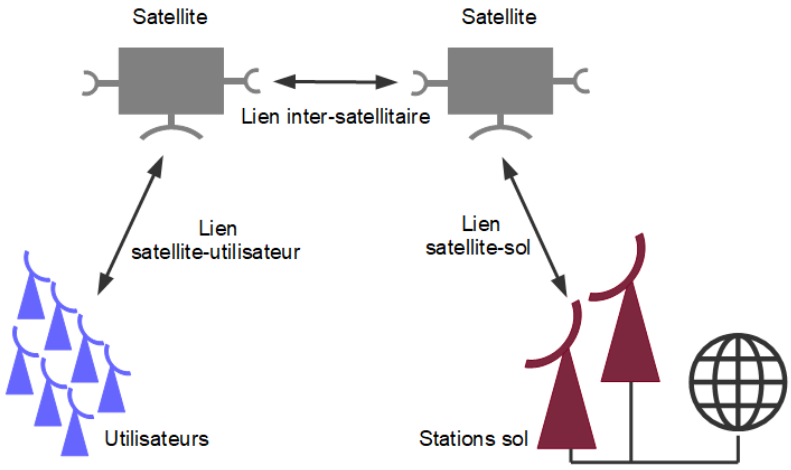


FIGURE 1.1 – Illustration des différents composants de la constellation

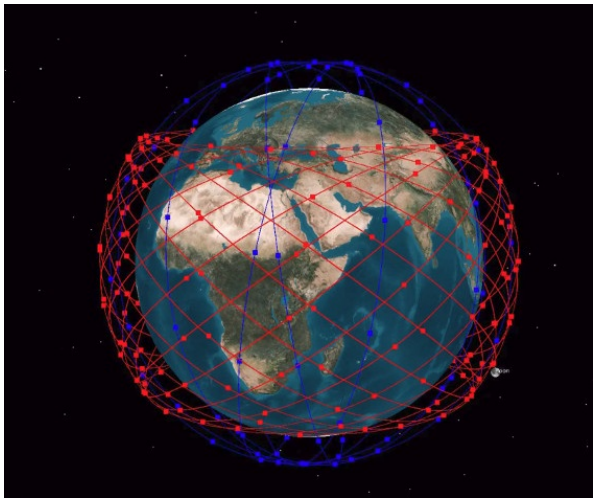


FIGURE 1.2 – Représentation de la constellation étudiée

Bien que la plupart des paramètres d'une constellation, tels que la dynamique ou les liens de télécommunication potentiels soient connus longtemps à l'avance, le système présente

tout de même une part d'aléatoire. En effet, la qualité des liens de communication entre les satellites et les objets au sol est fortement impactée par la météo et en particulier la présence de nuages. D'autre part, afin d'assurer un bon fonctionnement du système à long terme, il est nécessaire de prendre en compte les défaillances qui viendront dégrader les performances du système. Enfin, la demande Internet exacte de chaque utilisateur à chaque instant ne peut pas être connue à l'avance. Cependant, à part pour des défaillances permanentes, tous ces aléas devront être traités au dernier moment et ne sont pas pris en compte dans ce travail où nous nous intéressons à une planification des choix de télécommunications en amont de ces aléas. En effet, même la demande de chaque utilisateur peut être considérée comme en partie connue à l'avance car le débit de chaque utilisateur est régi par un contrat. Celui-ci stipule que l'opérateur est tenu de fournir un débit minimum à l'utilisateur, que ce dernier l'utilise ou non. Le contrat de chaque utilisateur étant connu à l'avance, le débit à réserver est connu en tout temps.

Le problème de la gestion des télécommunications dans une constellation de satellites se décompose en trois axes principaux : la gestion des liens satellite-utilisateur, la gestion des liens intersatellitaires et satellite-sol, et l'impact de la dynamique de la constellation sur les axes précédents. Dans la suite de ce chapitre, nous abordons ces trois axes avant de présenter une modélisation pour le problème complet. Au vu de la taille du problème, celui-ci ne peut être abordé frontalement et nous détaillons les simplifications faites en fin de chapitre.

1.1 Les liens satellite-utilisateur

Nous décrivons dans cette section les caractéristiques des liens satellites-utilisateurs impactant le problème de gestion de ces liens. Bien que nous fassions une description contenant des détails techniques, cette partie du problème de gestion des communications de la constellation ne sera pas prise en compte après la fin de ce chapitre. Cette description est donc faite pour donner un aperçu général du problème de gestion des communications satellitaires et pour servir de bases à la modélisation de programmation linéaire en nombre entiers de la Section 1.4.

Afin de communiquer avec les utilisateurs, les satellites sont équipés d'antennes actives. Une antenne active est une parabole composée de centaines d'éléments rayonnants. Chacun d'entre eux est capable d'émettre un signal avec une certaine amplitude et une certaine phase. Lorsque plusieurs éléments rayonnants émettent le même signal en même temps, en fonction de l'amplitude et de la phase de chacun et en fonction du point dans l'espace considéré, les interférences entre les signaux peuvent être constructives ou destructives. Ainsi, en calibrant correctement l'amplitude et la phase de chaque élément rayonnant, l'antenne peut émettre un signal fort dans une direction précise (le faisceau principal) et faible dans presque toutes les autres directions. Le faisceau principal peut être choisi de n'importe quelle forme dans n'importe quelle direction et la transmission peut être effectuée à tout moment. C'est cette flexibilité qui fait le succès des antennes actives. Cependant, dans certaines directions annexes, l'émission est assez forte pour créer des interférences non négligeables pour les autres signaux.

Malgré la grande flexibilité des antennes actives, certains liens entre les satellites et les utilisateurs ne peuvent pas être créés. Tout d'abord, un satellite ne peut créer un lien qu'avec les utilisateurs en visibilité. De plus, l'angle que forme le lien avec la verticale dans le référentiel du satellite ne doit pas être trop grand pour deux raisons. Premièrement, les éléments rayonnants d'une antenne émettent avec plus de puissance en face de l'antenne ce qui veut dire qu'un faisceau créé avec un grand angle par rapport à la verticale sera moins puissant et donc moins efficace. Deuxièmement, un faisceau éloigné de la verticale doit parcourir une plus grande distance dans l'air pour atteindre son utilisateur. Du fait de la dispersion de l'air, chaque mètre parcouru dans l'air réduit la puissance reçue par l'utilisateur et donc l'efficacité du lien. Pour ces deux raisons, les satellites uniquement accessibles à travers un lien de mauvaise qualité sont considérés comme n'étant pas en visibilité.

La principale décision à prendre pour les liens satellite-utilisateur est le choix du plan temps/fréquence des différents satellites. Plus précisément, parmi l'ensemble des utilisateurs, il faut choisir ceux vers lesquels chaque satellite créera un faisceau à chaque instant. Un satellite a la capacité de produire plusieurs faisceaux simultanément, mais aussi de partager ses ressources dans le temps à l'aide d'un cycle de *beam hopping*. Le cycle se compose de 16 slots (intervalles de temps de quelques millisecondes) et au cours de chaque intervalle, 24 faisceaux peuvent être créés. À la fin du cycle, le cycle suivant commence en utilisant la même séquence de faisceaux que précédemment. Par conséquent, un satellite possède 24×16 ressources à distribuer entre ces utilisateurs en visibilité. Ces ressources doivent être allouées avec soin. Tout d'abord, il faut s'assurer que chaque utilisateur puisse transmettre la quantité d'informations requise par son contrat. De plus, deux signaux émis dans les mêmes conditions (temps, fréquence, direction et polarisation) interfèrent, ce qui impacte le taux de transmission du signal.

Le débit fourni à un utilisateur dépend linéairement de trois paramètres : la bande passante b allouée à l'utilisateur, la proportion de temps τ durant laquelle la fréquence allouée est utilisée pour transmettre à l'utilisateur et l'efficacité spectrale η qui sert de coefficient transformant une bande passante en bits par seconde. Le débit de transmission est $R = \eta\tau b$. L'efficacité spectrale η dépend elle-même du système de modulation / encodage / redondance utilisé (MODCOD). Un MODCOD correspond à une modulation (choix d'une forme d'onde) et à un encodage de l'information (choix d'un code correcteur). Cependant, un MODCOD ne peut être utilisé que dans certaines conditions, un meilleur MODCOD nécessitant de meilleures conditions de transmission. Plus précisément, chaque MODCOD nécessite un rapport signal sur bruit minimum pour être efficace. Ce rapport, noté C/(N+I), est donné en dB et représente le niveau de bruit du signal émis. Pour chaque rapport signal sur bruit, le MODCOD utilisable d'efficacité spectrale maximale est sélectionné pour effectuer la transmission du signal. Cette correspondance entre rapport signal sur bruit et efficacité spectrale est illustrée en Figure 1.3 pour les protocoles DVB-S2 et DVB-S2x. Le rapport signal sur bruit d'un signal destiné à un utilisateur u est calculé à l'aide de la formule suivante :

$$\frac{C}{N+I} = \frac{P_u}{\sum_i P_i + I_{mod} + N}$$

Le numérateur P_u de la fraction est la puissance du signal utile en sortie d'antenne en direction de l'utilisateur u. Le premier terme du dénominateur est la somme des puissances émises par les autres faisceaux de l'antenne en direction de l'utilisateur u dans les mêmes conditions que le signal utile (temps, fréquence et polarisation). Le terme I_{mod} est la puissance du bruit introduit dans le signal lors de son amplification. Enfin, N est la puissance du bruit représentant la qualité des conditions à la réception (bruit capté par l'antenne et bruit interne au niveau du récepteur). Ce terme est transformé en un bruit équivalant à l'émission.

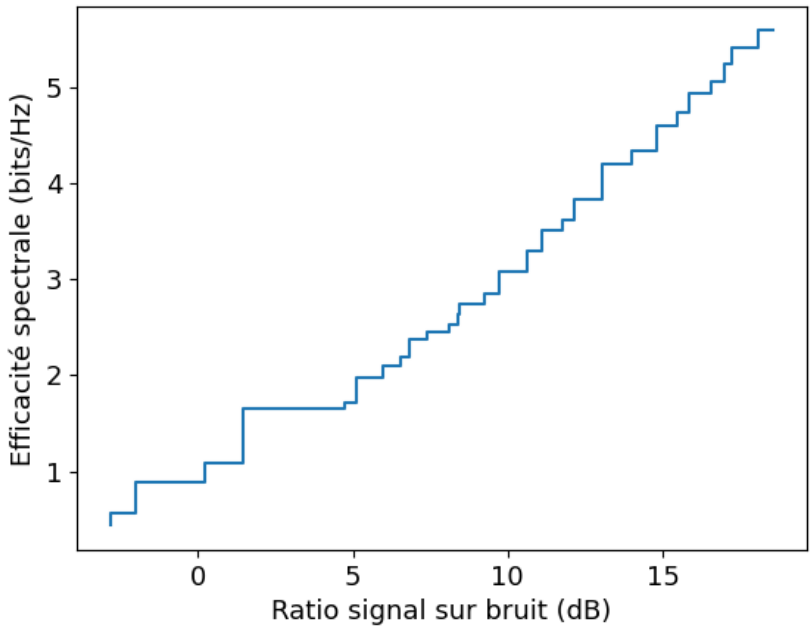


FIGURE 1.3 – Efficacité du meilleur MODCOD en fonction du rapport signal sur bruit pour le protocole DVB-S2x

L'objectif dans la gestion des liens satellite-utilisateur et donc dans le choix du plan temps/fréquence est de permettre à un maximum d'utilisateurs d'obtenir le débit requis dans son contrat. Ce débit est influencé par le nombre de ressources affectées à l'utilisateur et par le taux d'interférences présent sur ces ressources.

1.2 Les liens inter-satellitaires et satellite-sol : le problème de transmission

Une fois que chaque utilisateur est connecté à un satellite de la constellation, son flux Internet doit être acheminé vers son fournisseur d'accès Internet. Pour cela, le flux utilise trois types de connexions bidirectionnelles : des liaisons inter-satellites qui permettent au flux de se déplacer entre les satellites de la constellation, des liens satellite-sol qui connectent la constellation à des stations au sol et des liens fournisseur qui connectent ces mêmes stations

sol aux fournisseurs Internet. Les liaisons inter-satellites et les liens satellite-sol ne peuvent transférer que 10 Gbps de données Internet tandis que les liens fournisseurs qui font partie intégrante du réseau terrestre sont supposés avoir une capacité illimitée.

Comme mentionné dans l'introduction de ce chapitre, les satellites de la constellation Telesat sont connectés selon deux modèles de grilles toriques ne communiquant pas entre elles. En plus de ses quatre voisins immédiats sur la grille, chaque satellite peut se connecter à au plus quatre stations sol à la fois, à condition que ces stations sol soient visibles depuis le satellite.

L'un des points clés du problème de transmission du flux Internet des utilisateurs est que chaque utilisateur ne doit utiliser qu'un seul chemin en même temps pour acheminer son flux. En effet, un flux Internet est une séquence de blocs de données. Utiliser plusieurs chemins signifie répartir ces blocs sur les différents chemins. Or, l'ordre des blocs de données dans la séquence est essentiel à la compréhension des informations qu'ils contiennent. Pour ne pas avoir à utiliser un mécanisme de ré-ordonnancement, il faut s'assurer que les blocs arrivent à destination dans leur ordre d'émission. Ce fait est contradictoire avec l'utilisation de plusieurs chemins pour un même utilisateur. En effet, les connexions entre satellites, stations sol et fournisseurs d'accès Internet ont chacune une durée de transfert spécifique qui dépend de la distance entre les objets et du temps de traitement avant et après la transmission. Ainsi, chaque chemin dans la constellation a une durée de transfert différente et des blocs de données utilisant différents chemins pourraient ne pas arriver dans l'ordre dans lequel ils ont été envoyés.

En plus de réussir à transmettre le flux Internet, le choix des chemins doit garantir une bonne qualité de service pour les utilisateurs. Le temps de transmission des divers liens de communication impacte cette qualité de service à travers la latence introduite dans les communications. Ainsi, pour certaines applications, comme les visioconférences, la latence des communications doit rester assez basse afin que la qualité du service soit optimale. D'un point de vue technique, cela impose de garantir une latence maximum par utilisateur ou de pénaliser les chemins induisant une trop grande latence.

En résumé, l'objectif dans le problème de transmission est de permettre à tous les utilisateurs d'obtenir leur débit sur un chemin unique de faible latence tout en s'assurant que la capacité des liens de télécommunication de la constellation ne soit pas dépassée.

1.3 La dynamique de la constellation : impact sur les autres problèmes

Contrairement à des satellites géostationnaires, les satellites de la constellation ne surplombent pas le même point de la Terre en tout temps. Cela signifie que les utilisateurs et les stations sol n'ont pas toujours les mêmes satellites en visibilité. Ce fait a plusieurs impacts sur les problèmes de gestion des communications mentionnés ci-dessus.

Tout d'abord, pour rendre compte des mouvements de la constellation, une discrétisation temporelle est effectuée et une séquence des problèmes précédents est considérée. Ces différents pas de temps ne sont pas indépendants les uns des autres. En effet, pour deux pas de temps consécutifs, utiliser le même chemin dans la constellation pour transmettre le flux d'un utilisateur est valorisé. De fait, lorsqu'un utilisateur change le chemin qu'il emprunte, une synchronisation entre l'émetteur et le récepteur doit avoir lieu au cours de laquelle aucune information ne peut être transmise. Cette synchronisation implique une perte en termes de débit et donc le nombre de changements de chemin doit être minimisé. Cette considération a un impact sur les deux problèmes. Dans le problème du choix de plan temps/fréquence, un utilisateur doit rester connecté au même satellite aussi longtemps que possible. Dans le problème de transmission, un utilisateur doit utiliser le même chemin aussi longtemps que possible.

Comme nous allons le voir dans la prochaine section où nous présentons une modélisation du problème de la constellation, cet aspect temporel multiplie le nombre de variables du problème ce qui le rend d'autant plus difficile.

1.4 Un programme linéaire en nombres entiers pour le problème complet

Nous présentons maintenant un modèle de programmation linéaire en nombres entiers modélisant l'ensemble des aspects du problème de la constellation décrit dans les sections précédentes. Voici les notations utilisées pour écrire le modèle mathématique :

Nous notons $G = (V, (E_t)_{t \in T})$ le graphe orienté dynamique représentant la constellation et ses liens de télécommunication, avec V son ensemble de nœuds et E_t son ensemble d'arcs au pas de temps t. Nous utiliserons les indices suivantes :

- $t \in T$ pour le temps,
- $s \in S$ pour les unités temporelles (slots) des cycles de *beam hopping*,
- $e \in E_t$ pour les arcs représentant les liens de télécommunications,
- $v \in V$ pour les sommets représentant les objets communicants (satellites, utilisateurs, station sol, fournisseur réseau),
- $k \in K$ pour les utilisateurs,
- $m \in M$ pour les MODCOD.

Nous utiliserons les variables suivantes :

- f_{et}^k indique si le chemin utilisé par l'utilisateur k au temps t passe par l'arc e,
- o_{et} représente le dépassement de capacité sur l'arc e au temps t,
- d_{etsm} représente le débit envoyé sur l'arc e (lien satellite-utilisateur) au temps t sur le slot s en utilisant le MODCOD m,
- z_{etsm} indique si le MODCOD m est utilisé sur l'arc e (lien satellite-utilisateur) au temps t sur le slot s,

— x_{ets} indique si l'arc e (lien satellite-utilisateur) est utilisé au temps t sur le slot s,

— n_t^k indique si le chemin de l'utilisateur k change entre les pas de temps t et $t+1$.

Nous utiliserons les constantes suivantes :

— d_t^k est la demande de l'utilisateur k au temps t,

— O_t^k est l'origine du flux de l'utilisateur k au temps t (son fournisseur internet),

— D_t^k est la destination du flux de l'utilisateur k au temps t (la position de l'utilisateur),

— c_e est la capacité de l'arc e (hors lien satellite-utilisateur),

— γ_m est le ratio signal sur bruit nécessaire pour utiliser le MODCOD m,

— η_m est l'efficacité spectrale du MODCOD m,

— $P_{ee't}$ est la puissance émise dans la direction de l'utilisateur situé au bout de l'arc e' du fait des émissions sur l'arc e au temps t,

— N_{et} est le bruit présent sur l'arc e au temps t,

— B_{ets} est la bande passante de l'arc e (lien satellite-utilisateur) au temps t sur le slot s,

— N_l est le nombre de liens que peut utiliser chaque satellite à chaque slot temporel (égal à 24 pour la constellation Telesat),

— l_{et} est la latence du lien e au temps t,

— l_{max} est la latence maximale autorisée pour une transmission.

Nous utilisons également la notation de Kronecker : δ_x^y vaut 1 si $x = y$ et 0 sinon. Les ensembles d'arcs entrants et sortants du nœud v au temps t seront notés respectivement $E_t^-(v)$ et $E_t^+(v)$. De plus, E_t^k représente l'ensemble des liens de communication connectant au temps t l'utilisateur k aux satellites qu'il a en visibilité : $E_t^k = E_t^+(D_t^k) \cup E_t^-(D_t^k)$. Enfin, pour un lien e connectant un satellite et un utilisateur k, nous notons $E(e)$ l'ensemble des liens connectant le satellite en question avec les utilisateurs en visibilité autres que l'utilisateur k.

Le problème de gestion des communications d'une constellation de satellites tel que décrit précédemment peut alors être décrit à l'aide du programme linéaire en variables mixtes

suivant.

$$\min f((o_{et})_{e\in E_t,t\in T}) + \sum_{t\in T}\sum_{k\in K} n_t^k \tag{1.1a}$$

tel que

$$\sum_{e\in E_t^+(v)} f_{et}^k - \sum_{e\in E_t^-(v)} f_{et}^k = \delta_v^{O_t^k} - \delta_v^{D_t^k} \qquad \forall k\in K,\ \forall v\in V,\ \forall t\in T, \tag{1.1b}$$

$$\sum_{k\in K} f_{et}^k d_t^k \le c_e + o_{et} \qquad \forall e\in E_t, \forall t\in T, \tag{1.1c}$$

$$\sum_{e\in E_t} f_{et}^k l_{et} \le l_{max} \qquad \forall k\in K, \forall t\in T, \tag{1.1d}$$

$$\sum_{s\in S}\sum_{m\in M} d_{etsm} = f_{et}^k d_t^k \qquad \forall e\in E_t^k, \forall t\in T, \forall k\in K \tag{1.1e}$$

$$d_{etsm} \le z_{etsm} d_t^k \qquad \forall e\in E_t^k, \forall t\in T, \forall k\in K, \forall s\in S, \forall m\in M \tag{1.1f}$$

$$\sum_{m\in M} z_{etsm} = 1 \qquad \forall e\in E_t^k, \forall t\in T, \forall k\in K, \forall s\in S \tag{1.1g}$$

$$z_{etsm} P_{eet} \gamma_m \le \sum_{e'\in E(e)} x_{e'ts} P_{e'et} + N_{et} \qquad \forall e\in E_t^k, \forall t\in T, \forall k\in K, \forall s\in S, \forall m\in M \tag{1.1h}$$

$$\sum_{m\in M} \frac{d_{etsm}}{\eta_m} \le x_{ets} B_{ets} \qquad \forall e\in E_t^k, \forall t\in T, \forall s\in S, \forall k\in K \tag{1.1i}$$

$$\sum_{e\in E_t} x_{ets} \le N_l \qquad \forall t\in T, \forall s\in S \tag{1.1j}$$

$$f_{et}^k - f_{e,t+1}^k \le n_t^k \qquad \forall t\in T, \forall e\in E_t, \forall k\in K \tag{1.1k}$$

$$f_{et}^k \in \{0,1\},\ x_{ets}\in\{0,1\},\ z_{etsm}\in\{0,1\} \tag{1.1l}$$

$$n_t^k \in \{0,1\},\ d_{etsm}\in\mathbb{R}^+,\ o_{et}\in\mathbb{R}^+,\ o_t\in\mathbb{R}^+ \tag{1.1m}$$

La fonction objectif (1.1a) de ce problème comprend deux termes :

— $f((o_{et})_{e\in E_t,t\in T})$ est une pénalisation linéaire par morceaux du dépassement de capacité sur les arcs,

— $\sum_{t\in T}\sum_{k\in K} n_t^k$ compte le nombre de pénalités payées à cause des changements de chemins des utilisateurs.

Les contraintes (1.1b)-(1.1d) sont les contraintes associées au problème de transmission :

— (1.1b) est la contrainte de conservation du flot ; elle implique que l'ensemble des arcs sélectionnés pour un utilisateur avec les variables f_{et}^k forme un chemin.

— (1.1c) est la contrainte de capacité des liens (hors liens satellites-utilisateur). Tout dépassement de capacité est stocké dans les variables o_{et}.

— (1.1d) est la contrainte s'assurant que les chemins choisis pour chaque utilisateur ont

bien une latence inférieure à l_{max}.

Les contraintes (1.1e)-(1.1j) sont les contraintes associées au choix du plan temps/fréquence :

— (1.1e) est la contrainte faisant le lien entre les variables d_{etsm} et f_{et}^{k} qui représentent le débit des utilisateurs dans le plan temps/fréquence et dans le problème de transmission respectivement.

— (1.1f) est la contrainte s'assurant que du débit n'est envoyé avec le MODCOD m que si ce MODCOD a été choisi.

— (1.1g) implique qu'un seul MODCOD est choisi pour chaque transmission.

— (1.1h) est la contrainte garantissant qu'un MODCOD n'est choisi que si le rapport signal sur bruit nécessaire à son utilisation est respecté.

— (1.1i) est la contrainte garantissant que la quantité de bande passante utilisée pour les transmissions sur un lien satellite-utilisateur n'excède pas la quantité de bande passante disponible.

— (1.1j) est la contrainte assurant qu'un satellite ne génère pas plus de liens vers les utilisateurs que le maximum autorisé (24 liens).

Enfin, la contrainte (1.1k) modélise l'aspect dynamique de la constellation en garantissant qu'une pénalité est payée lorsque qu'un utilisateur change de chemin entre deux pas de temps.

Dans notre cas d'étude, le nombre de liens satellite-utilisateur est supérieur à 10^5. Si plus de 10 pas de temps, 10 slots de *beam hopping* et 10 MODCOD sont considérés, le nombre de variables binaires z_{etsm} est supérieur à 10^8 ce qui très largement au dessus de la capacité de résolution de tous les solveurs actuels. De même on peut estimer le nombre de variables f_{et}^{k} à 10^7.

1.5 Conclusion : vers les flots insécables

En raison de la grande complexité du problème complet de la constellation, nous décidons de restreindre notre étude à une sous-partie du problème. Dans un premier temps, nous nous intéressons uniquement au problème de transmission sans prendre en compte la contrainte de latence. Cela implique que le plan temps/fréquence des liens satellite-utilisateur est considéré fixé comme une donnée du problème. De plus, la constellation n'est considérée qu'à un instant donné, sa dynamique n'est pas prise en compte. Dans ce cas, le problème de transmission est un problème connu de la littérature : le problème de flot insécable.

Le problème de flot insécable est une variante largement étudiée du problème de flot multi-commodités. Dans ce problème, un graphe (orienté ou non) est donné avec des capacités sur ses arcs. Un ensemble de commodités, chacune composée d'une origine, d'une destination et d'une demande, est également fournie. Chaque commodité doit acheminer sa demande de son origine à sa destination par un chemin unique. L'ensemble de ces flots doit garantir que la capacité des arcs n'est pas dépassée, ou du moins que la violation des capacités est minimisée. Dans notre application, le graphe représente la constellation et les commodités représentent les utilisateurs et leur demande en débit.

Dans un deuxième temps, nous décidons de réintroduire la dynamique de la constellation dans le problème. En particulier, le modèle considère alors plusieurs pas de temps ainsi que les pénalités de changement de chemin. Ces ajouts induisent une variante dynamique du problème de flot insécable. Une séquence de problèmes de flot insécable est donnée, représentant les différents pas de temps. Chaque pas de temps introduit quelques changements au problème : certaines commodités changent d'origine ou de destination, certains arcs sont ajoutés ou supprimés. Cette séquence de problèmes comporte deux objectifs contradictoires : 1) minimiser le débit dépassant les capacités, 2) les commodités doivent, si possible, utiliser le même chemin lors de pas de temps consécutifs (le nombre de changements de chemin au cours du temps est minimisé).

Dans la suite de ce manuscrit, nous nous intéresserons à la résolution de ces deux problèmes pour des instances de grande taille afin de pouvoir utiliser nos résultats dans cette application industrielle.

Chapitre 2

État de l'art

Résumé du chapitre

Dans ce chapitre, nous passons en revue l'état de l'art concernant les principaux travaux reliés aux sujets discutés dans ce manuscrit. Nous commençons par traiter du problème de flot insécable en présentant plusieurs formulations de programmation linéaire en nombres entiers. Puis, nous décrivons les méthodes de résolution connues pour ce problème, qu'elles soient basées sur ces formulations ou sur d'autres méthodes telles que des heuristiques ou des méta-heuristiques. Dans un troisième temps, nous traitons des principaux résultats connus pour des problèmes connexes au problème de flot insécable. Par la suite, nous discutons du problème de flot insécable dynamique en traitant la formulation proposée dans la littérature ainsi que de la méthode utilisée pour la résoudre. Enfin une dernière partie est consacrée au différentes approches connues pour résoudre la relaxation linéaire du problème de flot insécable : le problème de flot multi-commodités.

Avant de commencer ce chapitre, nous rappelons certaines notations présentées au début de ce manuscrit. Lorsque nous discuterons de flots insécables statiques, utiliserons les notations suivantes :

— $G = (V, E)$ est un graphe orienté ou non, avec V son ensemble de nœuds et E son ensemble d'arcs ;

— $L = (O^k, D^k, d^k)_{k \in K}$ est une liste de commodités définies par leur origine, leur destination et leur demande ;

— $(c_e)_{e \in E}$ sont des capacités sur les arcs.

Lorsque nous discuterons de flots insécables dynamiques nous utiliserons les notations suivantes :

— $G = (V, (E_t)_{t \in T})$ est un graphe dynamique orienté ou non orienté où V et E sont les ensembles de nœuds et d'arcs et où $(E_t)_{t \in T}$ sont les ensembles d'arcs autorisés à chaque pas de temps ;

— $((O^k_t, D^k_t, d^k_t)_{t \in T})_{k \in K}$ est un ensemble de commodités définies par leur origine, leur destination et leur demande à chaque pas de temps ;

— $(c_{et})_{e \in E_t, t \in T}$ sont des capacités sur les arcs du graphe dynamique à chaque pas de temps.

Nous utilisons aussi la notation de Kronecker : δ_x^y est égal à 1 si $x = y$ et à 0 sinon. Les ensembles des arcs entrants et sortants d'un nœud v sont notés $E^-(v)$ et $E^+(v)$ respectivement. Enfin, $c_{min} = \min_{e \in E} c_e$ and $d_{max} = \max_{k \in K} d^k$ sont la plus petite capacité d'un arc et la plus grande demande d'une commodité, respectivement.

2.1 Formulations pour le problème de flot insécable

Dans cette section, nous introduisons formellement le problème de flot insécable à l'aide de formulations de programmation linéaire en nombres entiers.

2.1.1 Fonctions objectif

Quatre fonctions objectifs différentes sont présentes dans la littérature pour les problèmes de flot insécable :

1. maximiser la demande servie ou le profit des commodités servies (Kolman, 2003) ;
2. minimiser le coût induit par l'utilisation des arcs par le flot des commodités (Barnhart *et al.*, 2000) ;
3. minimiser le nombre de tournées nécessaires (Aumann et Rabani, 1995) : dans cette métrique, les commodités sont réparties entre plusieurs tournées en plus d'être affectées à un chemin ; le flot de l'ensemble des commodités d'une tournée ne doit pas dépasser la capacité des arcs ;
4. minimiser la congestion du graphe (Martens et Skutella, 2006), soit le plus petit nombre Δ par lequel il faut multiplier toutes les capacités pour pouvoir acheminer toutes les commodités. On peut aussi définir la congestion d'un arc comme le rapport entre le débit sur cet arc et sa capacité.

Dans ce travail, nous nous concentrons sur la minimisation de la congestion car c'est l'objectif le plus proche de notre application.

La congestion est une métrique qui met l'accent sur les arcs de faible capacité. De plus, minimiser la congestion du graphe n'impose aucune restriction au flot passant par les arcs n'ayant pas une congestion maximale. En particulier, cette métrique n'incite pas à minimiser la congestion sur ces arcs. Ceci devient problématique lorsqu'un arc est largement plus encombré dans chaque solution que tout autre arc car il lève toutes restrictions pour les autres arcs. Pour éviter cela, nous introduisons un nouveau critère pour minimiser la violation des capacités des arcs. Nous utilisons le terme *dépassement de capacité* pour décrire la quantité (toujours positive) de flot o_e qui dépasse la capacité d'un arc e. Notre nouveau critère consiste à minimiser la somme des dépassements de capacité $\sum_{e \in E} o_e$. Notons que la congestion Δ n'est pas le dépassement de capacité maximum sur tous les arcs. Ce nouveau critère sera utilisé par la suite dans nos expériences afin de comparer les solutions renvoyées par nos algorithmes.

De plus, nous présentons les formulations dans les sections suivantes à l'aide du dépassement de capacité. Il est possible de créer des formulations utilisant la congestion comme fonction objectif en remplaçant dans chaque formulation $c_e + o_e$ par $c_e\Delta$ et en minimisant Δ au lieu de $\sum_e o_e$. On peut noter que la variable Δ est commune à tous les arcs alors qu'il y a une variable o_e pour chaque arc dans les formulations utilisant le dépassement de capacité.

2.1.2 La formulation arc-nœud

La formulation arc-nœud que l'on peut trouver dans (Alvelos et De Carvalho, 2003) est une formulation compacte. En effet, elle comporte un nombre polynomial de variables et de contraintes en son nombre de commodités, de nœuds et d'arcs. Elle peut ainsi être résolue directement dans un solveur MILP pour des instances de petite taille. Cette formulation est caractérisée par ses contraintes de conservation du flot et contient les variables suivantes :

— f_e^k indique si la commodité k utilise l'arc e dans son chemin,

— o_e représente le dépassement de capacité sur l'arc e.

Le problème de flot insécable se formule alors :

$$\min_{f_e^k, o_e} \sum_{e \in E} o_e \tag{2.1a}$$

tel que

$$\sum_{e \in E^+(v)} f_e^k - \sum_{e \in E^-(v)} f_e^k = \delta_v^{O^k} - \delta_v^{D^k} \qquad \forall k \in K,\ \forall v \in V, \tag{2.1b}$$

$$\sum_{k \in K} f_e^k d^k \leq c_e + o_e \qquad \forall e \in E, \tag{2.1c}$$

$$f_e^k \in \{0,1\},\ o_e \in \mathbb{R}^+ \qquad \forall k \in K,\ \forall e \in E. \tag{2.1d}$$

L'équation (2.1b) est la contrainte de conservation du flot. Elle assure l'égalité des quantités de flot entrantes et sortantes en chaque nœud qui n'est ni l'origine ni la destination d'une commodité. L'équation (2.1c) est la contrainte de capacité. Elle assure que tous les dépassements de capacité sont bien stockés dans les variables o_e. Le fait que $f_e^k \in \{0,1\}$ implique que le flot est insécable.

2.1.3 La formulation par chemins

La formulation par chemins peut être obtenue en appliquant une décomposition de Dantzig-Wolfe à la formulation arc-nœud sur les contraintes de conservation du flot (Alvelos et De Carvalho, 2003). Celles-ci ne sont alors plus présentes dans la formulation et les variables représentent les chemins utilisés par chaque commodité. Cette formulation possède un nombre exponentiel de variables (en le nombre de nœuds et d'arcs) et doit être résolue par une tech-

nique MILP particulière nommée *Branch and Price* (Barnhart *et al.*, 2000). La signification des variables dans cette formulation est la suivante :

— x_p^k indique si la commodité k utilise le chemin p,

— o_e représente le dépassement de capacité sur l'arc e.

Le problème de flot insécable se formule alors :

$$\min_{x_p^k, o_e} \sum_{e \in E} o_e \tag{2.2a}$$

tel que

$$\sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K, \tag{2.2b}$$

$$\sum_{k \in K} \sum_{p \in P^k | e \in p} x_p^k d^k \leq c_e + o_e \qquad \forall e \in E, \tag{2.2c}$$

$$x_p^k \in \{0,1\},\ o_e \in \mathbb{R}^+ \qquad \forall p \in \bigcup_k P^k,\ \forall k \in K,\ \forall e \in E. \tag{2.2d}$$

Dans cette formulation, P^k est l'ensemble des chemins utilisables par la commodité k. L'équation (2.2b) assure qu'exactement un chemin est utilisé par chaque commodité. L'équation (2.2c) est la contrainte de capacité. Elle assure que tous les dépassements de capacité sont bien stocké dans les variables o_e. Le fait que $x_p^k \in \{0,1\}$ implique que le flot est insécable.

2.1.4 La formulation par patron de commodités

La formulation par chemins peut être obtenue en appliquant une décomposition de Dantzig-Wolfe à la formulation arc-nœud sur les contraintes de capacité. Les nouvelles variables représentent les patrons de commodités autorisés sur chaque arc. Cette formulation a l'avantage d'avoir une relaxation linéaire plus forte que les deux formulations précédentes mais possède un nombre exponentiel de variables (en le nombre de commodités). La résolution de cette formulation à l'aide d'une technique de *Branch and Price* a été étudiée par Alvelos et De Carvalho (2003). La signification des variables dans cette formulation est la suivante :

— y_e^g indique si le patron de commodités g est autorisé sur l'arc e. Si ce patron est autorisé, il implique un dépassement de capacité o_e^g.

Le problème de flot insécable se formule alors :

$$\min_{y_e^g, o_e} \sum_{e \in E} o_e \tag{2.3a}$$

tel que

$$\sum_{g \in G} y_e^g = 1 \qquad \forall e \in E \tag{2.3b}$$

$$\sum_{e \in \delta^+(v)} \sum_{g \in G | k \in g} y_e^g - \sum_{e \in \delta^-(v)} \sum_{g \in G | k \in g} y_e^g = \delta_v^{O^k} - \delta_v^{D^k} \qquad \forall k \in K,\ \forall v \in V \tag{2.3c}$$

$$\sum_{g \in G} y_e^g o_e^g \leq o_e \qquad \forall e \in E \tag{2.3d}$$

$$y_e^g \in \{0,1\},\ o_e \in \mathbb{R}^+ \qquad \forall e \in E,\ \forall g \in G \tag{2.3e}$$

Dans cette formulation $G = \{0,1\}^K$ est l'ensemble des patrons de commodités possibles. L'équation (2.3b) assure qu'exactement un patron de commodités est utilisé par chaque arc. L'équation (2.3c) est la contrainte de conservation du flot. Elle assure l'égalité des quantités de flot entrantes et sortantes en chaque nœud qui n'est ni l'origine ni la destination d'une commodité. Enfin, l'équation 2.3d assure que la variable o_e représente le dépassement de capacité de l'arc e. Le fait que $y_e^g \in \{0,1\}$ assure que le flot est insécable.

2.2 Méthodes de résolution pour le problème de flot insécable statique

Nous passons maintenant en revue les principales méthodes de résolution présentes dans la littérature pour le problème de flot insécable. Ces travaux sont regroupés en trois axes : méthodes exactes, algorithmes d'approximation et métaheuristiques. Un dernier groupe est consacré à la relaxation linéaire du problème de flot insécable (le problème de flot multi-commodités) dont la résolution est primordiale à la résolution du problème de flot insécable.

2.2.1 Méthodes exactes

Dans cette section, nous présentons les méthodes de résolution exacte proposées dans la littérature. On remarque que presque toutes les solutions envisagées sont basées sur la formulation par chemins et non sur la formulation arc-nœud. Ceci s'explique par le fait que la formulation arc-nœud possède un trop grand nombre de variables pour être utilisée sur des instances du problème de flot insécable de taille raisonnable. La formulation par chemins, quant à elle, contient moins de variables si elle résolue à l'aide d'une méthode de *Branch and Price*. Une des difficultés du problème de flot insécable est qu'il est non trivial de trouver une procédure de branchement efficace pour résoudre la formulation par chemins à l'aide

d'un *Branch and Price* (Alvelos et De Carvalho, 2003). En effet, l'application d'une décision de branchement demande dans certains cas de changer d'algorithme pour résoudre le sous-problème de *pricing* ce qui rend la procédure inefficace. Les premiers travaux se sont focalisés sur la recherche d'une procédure de branchement efficace.

Parker et Ryan (1993) ont utilisé la règle de branchement suivante dans leur *Branch and Price* : considérons une commodité k envoyant une proportion fractionnaire de son flot sur un chemin p dont les arcs sont $e_1, ..., e_{|p|}$. La règle de branchement proposée par Parker et Ryan (1993) crée $|p|+1$ branches. Dans chacune des $|p|$ premières branches, l'un des arcs du chemin p devient interdit tandis que sur la dernière branche, la commodité k est obligée d'utiliser le chemin p. Cette règle garde intacte la structure du sous-problème de *pricing* qui peut toujours être résolu à l'aide d'un algorithme de plus court chemin. Cependant, elle présente l'inconvénient majeur de créer un nombre important de branchements à chaque nœud.

Park *et al.* (1996) proposent d'utiliser la règle de branchement qui serait envisagée dans un solveur MILP : si une commodité envoie une proportion fractionnaire de son flot sur un chemin, soit ce chemin est interdit soit la commodité est obligée d'utiliser ce chemin. Cette règle de branchement crée uniquement deux branches à chaque nœud mais ne conserve pas la structure du problème de *pricing*. En effet, pour pouvoir continuer à générer des chemins lorsque certains sont interdits, Park *et al.* (1996) proposent de calculer les k-plus-courts chemins au lieu de calculer uniquement le plus court. Ce calcul est effectué à l'aide de l'algorithme de Yen (1971). Park *et al.* (1996) incluent également des coupes pour renforcer la relaxation linéaire de leur formulation. Ces coupes sont des *lifted cover cuts* appliquées aux contraintes de capacité. Ils mentionnent que ces coupes aident leur algorithme à trouver les solutions optimales ou à prouver l'optimalité des solutions trouvées plus rapidement.

Barnhart *et al.* (2000) présentent une procédure de *Branch and Price and Cut* appliquée à une formulation par chemins. La plupart des travaux ultérieurs utilisent leur méthode comme point de comparaison. Une contribution majeure de leur travail est leur stratégie de branchement. Contrairement aux stratégies de branchement précédentes, leur méthode maintient intacte la structure du sous-problème et crée uniquement deux branches à chaque nœud. Pour une commodité dans une solution non entière, le nœud de divergence est le premier nœud où le flot de la commodité se divise. Les arcs sortants du nœud de divergence sont divisés en deux sous-ensembles disjoints E_1 et E_2 contenant chacun au moins un des arcs utilisés par la commodité. La règle de branchement est d'interdire soit l'utilisation des arcs de E_1, soit l'utilisation des arcs de E_2. Dans les deux cas, la solution non entière précédente n'est plus valide et l'interdiction d'ensembles d'arcs maintient la structure du sous-problème intacte. Barnhart *et al.* (2000) ont également utilisé des *lifted cover cuts* pour renforcer les contraintes de capacité.

Les travaux suivant utilisent tous la règle de branchement de Barnhart *et al.* (2000) et focalisent leurs efforts sur le renforcement de la relaxation linéaire du problème de flot insécable au niveau des contraintes de capacité.

Park *et al.* (2003) ont hybridé la formulation par chemins et la formulation par patron de commodités. La relaxation linéaire de cette formulation est plus forte, ce qui diminue

le temps passé dans la procédure de branchement. Ils comparent aussi différentes règles de branchement, mentionnent que la règle la plus performante est celle de Barnhart *et al.* (2000) et ne présentent des résultats de calcul que pour celle-ci.

Belaidouni et Ben-Ameur (2007) ont présenté une méthode de plans sécants basée sur l'utilisation de fonctions super-additives afin de renforcer la relaxation linéaire de la formulation par chemin utilisée dans leur méthode *Branch and Price*. Il semble que sur de petites instances l'ajout de leurs coupes permette la création de solutions entières sans utiliser de procédure de *Branch and Bound*. Leur méthode est comparée à celle de Barnhart *et al.* (2000) et on constate une nette amélioration des résultats.

Les meilleurs résultats peuvent être trouvés dans les articles de Belaidouni et Ben-Ameur (2007) et Park *et al.* (2003). Belaidouni et Ben-Ameur (2007) ont comparé leur résultats à ceux de Barnhart *et al.* (2000) et résolvent toutes leurs instances (au plus 30 nœuds, 60 arcs, 100 commodités) en moins de 10 secondes. Park *et al.* (2003) n'ont comparé leurs résultats avec aucun des travaux précédents, mais résolvent des instances de même magnitude (au plus 30 nœuds, 80 arcs, 100 commodités) en moins de 15 secondes. Notons que les instances abordées dans ces travaux ont des ordres de grandeurs largement inférieurs à ceux visés pour les applications de nos travaux.

2.2.2 Algorithmes d'approximation et heuristiques

En tant que problème NP-difficile, beaucoup d'attention a été accordée aux algorithmes d'approximation et aux heuristiques résolvant le problème de flot insécable. Chaque algorithme d'approximation possède un facteur d'approximation λ. Soit Γ la fonction objectif du problème de minimisation en question. Les solutions S générées par un algorithme de facteur d'approximation λ vérifient l'inégalité suivante :

$$\Gamma(S^*) \leq \Gamma(S) \leq \lambda\Gamma(S^*),$$

où $\Gamma(S)$ et $\Gamma(S^*)$ sont respectivement la valeur de la solution générée et la valeur de la solution optimale. Cette inégalité garantit que le rapport entre la valeur de la solution produite et la valeur de la solution optimale n'est pas trop élevé. Nous renvoyons au *Handbook of Approximation Algorithms* (Gonzalez, 2020) pour une étude détaillée sur les algorithmes d'approximation dans le contexte des flots insécables. Nous rappelons ici quelques travaux liés à l'objectif de congestion minimum car c'est le plus proche de notre application et nous l'utiliserons lors des preuves d'approximations faites en Section 3.3.

Dans le contexte de la minimisation de la congestion, on cherche le plus petit nombre par lequel il est nécessaire de multiplier toutes les capacités afin de pouvoir affecter toutes les commodités. L'algorithme d'approximation le plus connu pour la congestion est la méthode de *Randomized Rounding* introduite par Raghavan et Tompson (1987) que nous appellerons l'algorithme RR. Cet algorithme possède un facteur d'approximation en $O(\frac{\ln|E|}{\ln\ln|E|})$. Cet borne supérieure est très proche de la borne inférieure en $\Omega(\frac{\ln|V|}{\ln\ln|V|})$ proposée par Chuzhoy

et al. (2007) pour les graphes orientés, en supposant $NP \nsubseteq BPTIME(|V|^{O(\ln\ln|V|)})$. De même, une borne en $\Omega(\ln\ln|V|/\ln\ln\ln|V|)$ est proposé par Andrews *et al.* (2010) pour le cas des graphes non orientés en supposant $NP \nsubseteq ZPTIME(|V|^{polylog(|V|)})$. Le processus de *Randomized Rounding* proposé par Raghavan et Tompson (1987) peut être dérandomisé en utilisant la méthode des probabilités conditionnelles (Young, 1995). Un algorithme de facteur d'approximation $(|K|+2)$ est présenté par Asano (2000). Les résultats numériques présentés montrent qu'en pratique cet algorithme se comporte de manière comparable à l'arrondi aléatoire classique.

Pour la fonction objectif de maximisation de la demande, le paramètre $\frac{d_{max}}{c_{min}}$ est introduit afin de mettre en valeur l'impact de la taille des commodités par rapport aux capacités des arcs sur la difficulté des instances. En effet, lorsque ce paramètre est borné par une petite valeur, plusieurs travaux rapportent des résultats d'approximation plus forts pour leurs algorithmes (Chakrabarti *et al.*, 2007; Shepherd et Vetta, 2015; Azar et Regev, 2006). Cependant, à notre connaissance, il n'existe pas de résultats similaires pour la fonction objectif de congestion.

En plus des algorithmes d'approximation mentionnés ci-dessus, quelques heuristiques, dont les propriétés d'approximation n'ont pas été étudiées, ont été proposées dans la littérature. Coudert et Rivano (2002) ont proposé un algorithme très similaire à l'algorithme SRR qui sera présenté en Section 3.2.1, sans prouver qu'il s'agit d'un algorithme d'approximation. Asano (2000) ainsi que Wang et Wang (1999) ont quant à eux proposé des heuristiques gloutonnes et d'autres basées sur la programmation linéaire. Les résultats rapportés montrent que les méthodes basées sur la programmation linéaire surpassent souvent les approches gloutonnes et donnent des résultats similaires à l'algorithme d'arrondi aléatoire de Raghavan et Tompson (1987).

2.2.3 Métaheuristiques

Il est NP-difficile de trouver une solution optimale à un facteur constant près pour le problème de flot insécable. C'est pourquoi la littérature a abordé ce problème sous l'angle des métaheuristiques. Parmi les algorithmes utilisés, on peut mentionner les algorithmes génétiques (Cox, 1991; Masri *et al.*, 2019), la recherche tabou (Anderson *et al.*, 1993; Laguna et Glover, 1993; Xu *et al.*, 1997), la recherche locale et la méthode GRASP (Alvelos et Valério de Carvalho, 2007; Santos *et al.*, 2010, 2013a,b; Masri *et al.*, 2015, 2019) ou l'optimisation par colonie de fourmis (Li *et al.*, 2010; Masri *et al.*, 2011). L'une des difficultés majeures rencontrées lors de la résolution du problème de flot insécable à l'aide d'une métaheuristique est de parcourir efficacement l'ensemble des chemins utiles pour chaque commodité. Nous détaillons maintenant les solutions proposées dans la littérature pour résoudre ce problème.

Les premières approches telles que (Cox, 1991; Anderson *et al.*, 1993) encodent les solutions comme des permutations de commodités. C'est cet espace de permutations qui est exploré par la métaheuristique. La création d'une solution se fait alors de la façon suivante. Dans l'ordre de la permutation, chaque commodité est allouée au chemin le plus court ayant suffisamment de capacité pour contenir la demande de la commodité. La solution est évaluée lorsque chaque

commodité a été affectée.

Dans (Laguna et Glover, 1993) et (Masri *et al.*, 2015), les k-plus-courts chemins sont précalculés pour chaque commodité en utilisant l'algorithme de Yen (1971). L'espace de recherche de leurs métaheuristiques est restreint à l'espace des solutions utilisant uniquement ces chemins.

Une autre idée proposée par Alvelos et Valério de Carvalho (2007) et Santos *et al.* (2010, 2013a,b) est d'utiliser l'information obtenue lors de la résolution de la relaxation linéaire du problème. La relaxation de la formulation par chemins est résolue avec un algorithme de génération de colonnes au cours duquel un ensemble de chemins est généré pour chaque commodité. Une métaheuristique est ensuite utilisée pour explorer les solutions où seuls ces chemins sont utilisés. Dans (Santos *et al.*, 2013b), après la résolution de la première relaxation linéaire, des modèles linéaires perturbés sont résolus pour créer de nouvelles colonnes et étendre l'espace de recherche de la métaheuristique.

L'optimisation par colonies de fourmis est également un moyen de naviguer au sein du vaste ensemble des chemins possibles (Li *et al.*, 2010; Masri *et al.*, 2011). En effet, à chaque itération d'un algorithme de colonies de fourmis, chaque commodité crée un chemin en prenant en compte plusieurs métriques : la longueur du chemin, la congestion du chemin et les phéromones. Chaque arc possède un niveau de phéromones pour chaque commodité. Plus celui-ci est élevé, plus la probabilité que l'arc appartienne au chemin généré est élevée. Les phéromones sont mises à jour de deux manières. Premièrement, les meilleures solutions ajoutent des phéromones à l'arc qu'elles utilisent. Deuxièmement, les phéromones s'évaporent afin que leur niveau ne devienne pas excessif. Ceci facilite l'exploration de l'espace des solutions.

Nous renvoyons aux travaux de Li *et al.* (2010) et Santos *et al.* (2013a) pour les métaheuristiques les plus performantes. Li *et al.* (2010) ont comparé leurs résultats avec le solveur CPLEX et ont résolu des instances possédant jusqu'à 60 nœuds, 400 arcs et 3 500 commodités en moins de 900 secondes. Santos *et al.* (2013a) montrent que toutes leurs instances (26 nœuds, 80 arcs, 500 commodités) sont résolues en moins de 180 secondes avec des valeurs proches de la borne inférieure de relaxation linéaire. Ces instances possèdent des graphes de taille similaire à ceux utilisés pour tester les méthodes de résolution exacte en revanche elles possèdent un nombre de commodité largement supérieur.

2.3 Problèmes similaires au problème de flot insécable statique

Dans cette section, nous présentons quelques travaux reliés à des problèmes proches du problème de flot insécable.

Flots insécables mono-source : le problème de flot insécable mono-source est un problème de flot insécable dans lequel toutes les commodités partagent la même source ou de manière équivalente le même puits. L'intérêt de cette variante n'est pas dans les solutions

exactes et dans les métaheuristiques pour lesquelles les méthodes utilisées sont celles proposée pour le problème de flot insécable classique. En revanche, le problème de flot insécable mono-source ne possède pas les mêmes propriétés d'approximation que le cas général lorsque la fonction objectif est la congestion (Chuzhoy *et al.*, 2007). Tandis que le meilleur facteur d'approximation pour le cas général a été montré comme étant égale à $\Omega(\frac{\ln|V|}{\ln\ln|V|})$, Dinitz *et al.* (1999) ont proposé un algorithme d'approximation de facteur 2 pour le cas mono-source lorsque la plus grande demande est inférieure à la plus petite capacité. Cet algorithme donne de très bons résultats en pratique lorsque la plus grande demande des commodités est petite devant la plus petite capacité des arcs. D'autres algorithmes d'approximation plus anciens peuvent être trouvés dans (Kleinberg, 1996; Kolliopoulos et Stein, 1997). Plus récemment, Peng *et al.* (2007) ont étudié les propriétés d'approximation du problème de trouver le flot insécable vérifiant les capacités des arcs et de coût minimum. Dans ce cas, ils présentent un algorithme capable de renvoyer une solution de congestion 2 et dont le coût est inférieur au coût du meilleur flot vérifiant les capacités des arcs. Ce résultat, combiné au fait qu'il est NP-difficile de trouver une solution ayant une congestion inférieure $2-\epsilon$ et un coût inférieur à celui du meilleur flot vérifiant les capacités des arcs (Erlebach et Hall, 2004), clôture une partie des discussions sur l'approximabilité du problème de flot insécable mono-source. D'autres travaux sur le problème de flot insécable mono-source à coût minimum sont présentés par Skutella (2002); Martens *et al.* (2007).

Flots k-sécables : le problème des flots k-sécables, introduit par Baier *et al.* (2002), étudie un problème de flot où chaque commodité est autorisée à utiliser k chemins. Ce problème apparaît dans plusieurs applications de télécommunication. En effet, l'intérêt des flots k-sécable est d'obtenir de meilleures solutions en terme de valeur de la fonction objectif tout en utilisant un faible nombre de chemins pour chaque commodité. Le problème des flots k-sécables est plus difficile que celui des flots insécables. La plupart des résultats théoriques et des algorithmes d'approximation pour les flots k-sécables sont donnés dans (Baier *et al.*, 2002; Martens et Skutella, 2006; Koch et Spenke, 2006; Białoń, 2017). Deux des résultats les plus notables sont les suivants. Lorsque la valeur de k est constante, les flots k-sécables sont NP-difficiles à approximer avec un facteur de $\frac{5}{6}$. D'autre part, les flots k-sécables sont NP-difficiles pour $2 \leq k \leq |E| - |V| + 1$ et polynomiaux sinon. De nombreux travaux ont été effectués sur des méthodes exactes pour ce problème et on recense quatre principaux modèles de programmation linéaire en nombres entiers. Caramia et Sgalambro (2008) proposent une formulation arc-nœud avec des règles de branchement spéciales. Truffot *et al.* (2005) utilisent une formulation par chemins où chacun des k chemins de chaque commodité possède son propre ensemble de variables. Gamst *et al.* (2010) proposent une formulation où tous les chemins d'une commodité partagent le même ensemble de variables. Enfin, Gamst (2013) propose d'utiliser des variables décidant si une combinaison de chemins est utilisée. La meilleure solution MILP est basée sur une formulation où tous les chemins d'une commodité partagent le même ensemble de variables et est étudiée par Gamst et Petersen (2012). D'autres solutions sont présentées par Truffot et Duhamel (2008); Truffot *et al.* (2010). Enfin, des heuristiques et métaheuristiques sont étudiées par Caramia et Sgalambro (2010); Gamst (2014); Jiao *et al.* (2014, 2015).

***Network design* avec flot insécable :** le *network design* est une vaste catégorie de

problèmes étendant les problèmes de flots. Dans ces problèmes, en plus de choisir les chemins utilisés pour le flot des commodités, il est nécessaire de décider de la capacité des arcs du graphe sur lequel évoluent ces commodités. Ce choix de capacité peut représenter l'ouverture ou la fermeture d'arc mais aussi l'agrandissement de la capacité des liens d'un réseau existant. De plus, ces changements de capacités peuvent être continus ou discrets, bornés ou non, et induisent des coûts pouvant prendre diverses formes. En effet, ceux ci peuvent être proportionnels au changement de capacité ou complètement non linéaires. Nous intéresseront ici uniquement au cas où le flot à transmettre dans le graphe est insécable, c'est à dire composé d'un seul chemin par commodité. Le problème *network design* avec flot insécable a été étudié sous plusieurs angles dans la littérature. Ainsi, plusieurs travaux ont proposé des coupes pour les contraintes de capacité de ce problème qui ont ensuite été utilisées dans des méthodes de *Branch and Cut* afin d'accélérer leur convergence (Atamtürk et Rajan, 2002; Benhamiche *et al.*, 2016, 2020). De plus, Chen *et al.* (2021) ont appliqué une décomposition de Fenchel sur ce problème qui est une méthode génération exacte de coupe présentée en détails dans le Chapitre 5 de ce manuscrit. Enfin, en plus de méthodes exactes, ce problème a été étudié sous l'angle des métaheuristiques par Yaghini et Akhavan (2012) qui proposent un algorithme de recuit simulé.

2.4 Les flots insécables dynamiques

Le problème de flot insécable dynamique est une variante du problème de flot insécable où plusieurs pas de temps sont considérés. Cette variante dynamique permet de prendre en compte le mouvement de la constellation autour de la terre et possède deux objectifs contradictoires : respecter la capacité de chaque arc et minimiser le nombre de fois que chaque commodité change de chemin. Par le passé, ce problème a été étudié par Gamvros et Raghavan (2012) qui ont proposé une formulation de programmation linéaire en nombres entiers. Cette formulation est résolue à l'aide d'un algorithme de *Branch and Price and Cut*. Une de leurs contributions majeures est leur méthode de résolution du problème de *pricing* permettant de générer les variables de la formulation proposée. Une description approfondie de cette méthode est donnée en Section 4.2.1. D'autres problèmes ont été étudié dans leur variante dynamique. C'est ainsi le cas du *network design* où l'on retrouve aussi des approches à plusieurs pas de temps (Contreras *et al.*, 2011; Lee et Dong, 2009; Fragkos *et al.*, 2017). Dans ces approches, il existe un coût pour faire changer d'état (ouvert/fermé) les composants d'un réseau que ce soient des arcs ou des nœuds. Ces coûts sont très similaires aux pénalités de changement de chemin étudiées dans le problème de flot insécable dynamique. D'autre part, le problème de tournée de véhicules possède une variante où servir les utilisateurs en utilisant toujours les mêmes véhicules est valorisé (Kovacs *et al.*, 2014; Luo *et al.*, 2015; Stavropoulou *et al.*, 2019). Cette constance du véhicule dans le temps rappelle l'incitation à garder le même chemin dans le problème de flot insécable dynamique.

Nous présentons maintenant une formulation inspirée de la formulation du problème de flot insécable dynamique étudiée par Gamvros et Raghavan (2012). L'unique différence est que la fonction objectif utilisée dans la formulation présenté ci-dessous comprend une pénalisation du

dépassement de capacité. En effet, afin permettre au solveur d'avoir une certaine flexibilité, le choix des chemins de l'ensemble des commodités est autorisés, à chaque pas de temps, à induire une certaine quantité B de dépassement sans encourir de pénalisation. Passé ce montant, le dépassement de capacité est fortement pénalisé. Quant aux changements de chemin, ceux-ci sont modélisés avec des pénalités dont la valeur est unitaire quelque soit la commodité et le pas de temps considéré. Le nombre de variables de cette formulation n'est polynomial ni en le nombre de nœuds/arcs ni en le nombre de pas de temps et sa relaxation linéaire doit être résolue à l'aide d'une génération de colonnes. Cependant, sa structure est simple ce qui peut lui permettre de bien se comporter sur de grandes instances. Dans cette formulation, la signification des variables est la suivante :

— x_s^k décide si la commodité k utilise la séquence de chemins s. Si une séquence $(p_1, ..., p_{|T|})$ est utilisée, alors le chemin p_t est utilisé au pas de temps t. Chaque séquence de chemins induit un nombre n_s^k de changements de chemin.

— o_{et} représente le dépassement de la capacité de l'arc e au pas de temps t

— o_t représente le montant du dépassement qui excède, au pas de temps t, le montant B de dépassement non pénalisé. Ce montant est pénalisé avec un facteur de pénalisation α considérer unitaire dans le reste de ce manuscrit.

Nous notons S^k l'ensemble des séquences de chemins utilisables par la commodité k tandis que S_{et}^k désigne l'ensemble des séquences de chemins utilisables par la commodité k pour lesquelles le chemin du pas de temps t passe par l'arc $e \in E_t$.

$$\min_{x_s^k, o_{et}, o_t} \sum_{k \in K} \sum_{s \in S^k} n_s^k \, x_s^k + \alpha \sum_{t \in T} o_t \tag{2.4a}$$

tel que

$$\sum_{s \in S^k} x_s^k = 1 \qquad \forall k \in K \tag{2.4b}$$

$$\sum_{k \in K} \sum_{s \in S_{et}^k} x_s^k \, d^k \leq c_e(1 + o_{et}) \qquad \forall e \in E_t \; \forall t \in T \tag{2.4c}$$

$$\sum_{e \in E_t} o_{et} \leq B + o_t \qquad \forall t \in T \tag{2.4d}$$

$$x_s^k \in \{0,1\}, \; o_{et} \in \mathbb{R}^+, \; o_t \in \mathbb{R}^+ \qquad \forall s \in \bigcup_k S^k, \; \forall k \in K, \; \forall e \in E_t \forall t \in T \tag{2.4e}$$

L'équation (2.4b) garantit qu'exactement une séquence de chemins est choisie pour chaque commodité. Les équations (2.4c) et (2.4d) sont des contraintes de capacité : elles garantissent que o_{et} représente le dépassement de capacité sur l'arc e sur le pas de temps t et que o_t représente le montant de dépassement qui dépasse un seuil B au pas de temps t. Le fait que $x_s^k \in \{0,1\}$ garantit que le flot est insécable.

2.5 Relaxation linéaire des flots insécables : le problème de flot multi-commodité

Le problème de flot multi-commodité est la relaxation linéaire du problème de flot insécable. La valeur de sa solution optimale est une borne inférieure de la valeur optimale du problème de flot insécable. De ce fait, cette relaxation linéaire est utilisée dans plusieurs méthodes exactes et d'approximation. En tant que cas particulier de la programmation linéaire, le problème de flot multi-commodité peut être résolu en temps polynomial. Cependant, les méthodes basées sur des modèles simples de programmation linéaire ne sont pas toujours assez performantes pour obtenir une solution exacte en un temps restreint. C'est pourquoi la littérature à largement étudié ce problème.

Beaucoup d'efforts ont été investis dans des méthodes exactes pour ce problème. Même si un solveur commercial peut résoudre la formulation arc-nœud, cette méthode peut prendre un temps prohibitif pour les grandes instances. Une solution alternative est d'effectuer une relaxation Lagrangienne des contraintes de capacité pour décomposer le problème en sous-problèmes plus faciles (Retvdri *et al.*, 2004). Comme le rapportent Dai *et al.* (2017), la relaxation Lagrangienne est moins performante dans la plupart des cas que d'appliquer un algorithme de génération de colonnes à la formulation par chemin. Cependant, la relaxation Lagrangienne semble être le meilleur choix lorsque le nombre de commodités est très grand car son temps de calcul dépend linéairement de ce paramètre. Dans la plupart des autres cas, la génération de colonnes semble être la meilleure solution.

Plusieurs travaux ont contribué à augmenter les performances des algorithmes de génération de colonnes pour ce problème. Tout d'abord, une génération de colonnes primale-duale est présentée par Gondzio *et al.* (2013); Gondzio et González-Brevis (2015); Gondzio *et al.* (2016). Cette méthode consiste à utiliser un algorithme de points intérieurs pour résoudre le problème maître et ainsi à obtenir des solutions sous-optimales mais centrées (n'étant pas sur la frontière du polyèdre des contraintes). Ces solutions centrées sont utilisées pour calculer de nouvelles colonnes dans le sous-problème, ce qui stabilise le processus de génération de colonnes et réduit le nombre d'itérations nécessaires pour converger. Une autre approche, qui pourrait être combinée avec la précédente, est l'utilisation de variables agrégées présentée par Bauguion *et al.* (2013, 2015). Dans cette méthode, les variables ne représentent pas des chemins mais des chemins agrégés tels que des arbres ou des structures plus complexes. Les auteurs rapportent que les sous-problèmes associés aux variables agrégées peuvent être résolus efficacement. Les variables agrégées réduisent la taille du problème principal pendant l'algorithme mais peuvent induire un plus grand nombre d'itérations. L'agrégation doit donc être effectuée avec soin. Babonneau *et al.* (2006) présentent une méthode de points intérieurs spécialisée pour résoudre le problème de flot multi-commodité. Cette méthode est améliorée par Castro et Cuesta (2012). D'autres contributions aux méthodes de programmation linéaire peuvent être trouvées dans (Moradi *et al.*, 2015; Dai *et al.*, 2016a,b)

Pour les grandes instances, les méthodes de programmation linéaire peuvent prendre beaucoup de temps de calcul avant de trouver la solution optimale. Ainsi, la littérature s'est

intéressée à la recherche d'algorithmes d'approximation combinatoires et en particulier aux schémas d'approximation entièrement en temps polynomial (FPTAS en anglais). Les meilleurs résultats sont obtenus grâce à l'affectation de longueurs exponentielles aux arcs du graphe. Cette idée a été introduite pour la première fois par Shahrokhi et Matula (1990). Dans leur algorithme, une longueur exponentielle en le flot présent sur l'arc est attribuée à chaque arc. Le flot est augmenté de manière itérative sur le chemin le plus court reliant l'une des paires source-puits. Leur algorithme a été amélioré par Fleischer (2000) qui a montré que seul le calcul d'un plus court chemin approximatif était nécessaire. L'algorithme de Fleischer (2000) est le FPTAS le plus rapide en pratique sans être celui avec la plus petite complexité. En effet, l'algorithme de Madry (2010) possède une complexité inférieure mais Emanuelsson (2016) montre dans son travail que l'algorithme de Fleischer (2000) est plus rapide pour les instances ayant moins de 100 millions d'arcs.

2.6 Positionnement des contributions de la thèse par rapport à la littérature

Dans ce chapitre, nous avons vu que le problème de flot insécable a été largement étudié dans des travaux précédents qui l'ont abordé sous plusieurs angles. En particulier, de nombreux travaux ont proposé des méthodes de résolution exacte. De plus ses propriétés en terme d'approximabilité sont connues même s'il reste certains cas où l'écart entre les bornes supérieures et inférieures n'est pas encore comblé. En revanche, l'étude des instances de grande taille n'a pour le moment pas reçu une très grande attention. La plus grande instance traitée dans la littérature possède 100 nœuds (Masri *et al.*, 2015). Dans notre première contribution, nous nous intéressons aux algorithmes capables de donner des solutions de bonne qualité sur les grandes instances (typiquement 400 noeuds pour l'instance issue de la constellation Telesat). Ainsi, nous proposons une heuristique étendant l'algorithme RR de Raghavan et Tompson (1987) et donnant de bons résultats pour ces instances. L'efficacité de cette heuristique s'accroît lorsque les commodités ont une faible demande par rapport aux capacités des arcs. Cette propriété a été montrée pour certains algorithmes d'approximation étudiant d'autres fonctions objectif que la congestion. Nous étendons les justifications théoriques de cette propriété pour les algorithmes d'arrondi aléatoire tels que notre heuristique dans le cadre de la congestion.

D'autre part, nous avons vu que le problème de flot insécable dynamique n'a, quant à lui, pas reçu une très grande attention. Dans notre deuxième contribution, nous élargissons l'ensemble des algorithmes testés sur ce problème en concevant plusieurs méthodes basés sur diverses formulations du problème de flot insécable dynamique ou de sa relaxation linéaire. Dans le but de résoudre des instances de grande taille, nous proposons principalement des heuristiques qui viennent s'ajouter à la méthode exacte proposée par Gamvros et Raghavan (2012). Par ailleurs, nous présentons de nouvelles approches pour résoudre le sous-problème de la génération de colonnes de la formulation par séquences de chemins proposée par Gamvros et Raghavan (2012). Ces méthodes ne possèdent pas les limitations de la méthode proposée par Gamvros et Raghavan (2012) et possèdent une meilleure complexité dans le pire cas.

Enfin, notre troisième contribution porte sur les méthodes de décomposition en programmation linéaire en nombres entiers. En effet, la plupart des méthodes de résolution du problème de flot insécable utilisent une relaxation linéaire. Nous nous intéressons donc au renforcement de cette relaxation linéaire à l'aide de méthodes de décomposition. Une synthèse des principales méthodes existantes pour les décompositions de Dantzig-Wolfe et de Fenchel est présentée dans le Chapitre 5. De plus, nous proposons une nouvelle méthode de décomposition empruntant des idées aux deux méthodes précédentes. Ces trois méthodes sont comparées empiriquement sur des instances du problème de flot insécable et la nouvelle méthode démontre de meilleures performances sur ces instances. De plus, nous proposons une nouvelle méthode de résolution pour le sous-problème de la décomposition de Fenchel lorsque la normalisation utilisée est la normalisation directionnelle.

Chapitre 3

Le problème de flot insécable

Résumé du chapitre

Dans ce chapitre, nous étudions des algorithmes capables de résoudre des instances de grande taille du problème de flot insécable. Nous présentons et analysons en détail une heuristique basée sur l'arrondi aléatoire et étendant l'algorithme de Raghavan et Tompson (1987). Nous fournissons des preuves empiriques que cette approche est compétitive avec les méthodes de résolution de l'état de l'art grâce à son temps de calcul inférieur et/ou à ses solutions de meilleure qualité. De plus, nous proposons une variante de cette heuristique possédant le même facteur d'approximation que le meilleur algorithme d'approximation de la littérature et proposons pour ces deux algorithmes d'approximation une analyse plus précise de leurs garanties d'approximation. Cette nouvelle analyse met en valeur l'impact de la taille des demandes des commodités relativement aux capacités des arcs sur les performances de ces algorithmes. Enfin, nous introduisons une nouvelle fonction objectif pour le problème de flot insécable et discutons de ses différences avec la fonction objectif de congestion classiquement utilisée.

Le problème de flot insécable est une variante NP-complète du problème de flot multi-commodités. Dans ce problème, un graphe orienté ou non est donné et des capacités sont affectées à ses arcs. Une famille de commodités, chacune composée d'une origine, d'une destination et d'une demande, est également fournie. Chaque commodité doit acheminer sa demande de son origine à sa destination par un chemin unique. L'ensemble de ces flots doit garantir que la capacité des arcs n'est pas dépassée, ou du moins que la violation des capacités est minimisée. Ce problème a été présenté plus en détails à l'aide de formulations linéaires en nombres entiers en Section 2.1.

Ce problème a diverses applications, notamment dans les réseaux de télécommunication (*e.g.* réseaux optiques, satellites de télécommunication) et le transport de marchandises. Dans l'application en télécommunications spatiales qui nous intéresse, des instances de grande taille apparaissent possédant jusqu'à 500 nœuds, 2 000 arcs et 150 000 commodités. Cependant, seules quelques méthodes dans la littérature peuvent s'adapter à de telles instances. C'est le cas de l'algorithme d'approximation de Raghavan et Tompson (1987) et de certaines méta-heuristiques réglées pour utiliser peu de temps de calcul. L'algorithme présenté par Raghavan et Tompson (1987) fait partie des heuristiques les plus efficaces pour les grandes instances du

problème de flot insécable. Cet algorithme possède un facteur d'approximation de $O\left(\frac{\ln|E|}{\ln\ln|E|}\right)$, ce qui est théoriquement le meilleur facteur d'approximation réalisable. En revanche, les solutions qu'il trouve sont souvent loin d'être optimales.

Dans ce chapitre, nous nous concentrons sur un algorithme donnant de bons résultats pratiques sur des instances de grande taille. Cet algorithme est une extension heuristique de l'algorithme d'arrondi aléatoire de Raghavan et Tompson (1987). En tant que tel, il utilise également un processus d'arrondi aléatoire sur la relaxation linéaire du problème de flot insécable pour créer une solution insécable. Cet algorithme alterne les étapes d'arrondis aléatoires et les résolutions de la relaxation linéaire et sera donc appelé, dans ce travail, *Sequential Randomized Rounding* (SRR). Cette heuristique est également une extension de l'algorithme proposé par Coudert et Rivano (2002) pour lequel aucune preuve complète du facteur d'approximation n'a été donnée. Par rapport à l'algorithme de Coudert et Rivano (2002), l'heuristique SRR offre plus de flexibilité sur le nombre de résolutions de la relaxation linéaire et, plus important encore, tire parti du fait que les commodités peuvent avoir des niveaux de demande différents.

Nous décrivons également une variante de l'heuristique SRR pour laquelle nous prouvons les mêmes garanties d'approximation que l'algorithme de Raghavan et Tompson (1987). Cette variation sera appelée *Constrained Sequential Randomized Rounding* (CSRR). De plus, nous améliorons l'analyse de CSRR et de l'algorithme de Raghavan et Tompson (1987) en montrant qu'ils atteignent un facteur d'approximation $O\left(\frac{\gamma\ln|E|}{\ln(\gamma\ln|E|)}\right)$ où γ est un paramètre qui est petit lorsque les demandes de commodités sont faibles par rapport aux capacités des arcs. Enfin, nous montrons expérimentalement que l'algorithme SRR est capable de résoudre de grandes instances. Nous montrons aussi que, sur de grandes instances, l'algorithme SRR renvoie de meilleures solutions que les autres algorithmes de la littérature. Afin de faire cette comparaison avec la classe des métaheuristiques, nous proposons un algorithme de recuit simulé qui surclasse les autres métaheuristiques de la littérature.

Ce chapitre est structuré comme suit. Nous présentons en détail en Section 3.1 l'algorithme de Raghavan et Tompson (1987). La Section 3.2 décrit l'heuristique SRR et son analyse de complexité. La Section 3.3 décrit l'algorithme CSRR et fournit l'analyse menant au facteur d'approximation $O\left(\frac{\gamma\ln|E|}{\ln(\gamma\ln|E|)}\right)$. En Section 3.4, nous fournissons des résultats expérimentaux qui comparent la qualité empirique des différents algorithmes présentés. Enfin, en Section 3.5, nous discutons des différentes propriétés de l'heuristique SRR et de l'algorithme CSRR.

3.1 L'algorithme de Raghavan et Thompson

Nous présentons maintenant l'algorithme d'approximation le plus connu pour minimiser la congestion et qui est à la base de l'algorithme que nous proposons dans les sections suivantes. Cet algorithme est une méthode d'arrondi aléatoire introduite par Raghavan et Tompson (1987) que nous appellerons l'algorithme *Randomized Rounding* (RR) dans ce travail. Cette méthode s'effectue en deux étapes. Tout d'abord, une solution de la relaxation linéaire du problème est calculée. Chaque commodité est autorisée à utiliser plusieurs chemins dans cette solution. Dans une formulation par chemins, la distribution du flot de chaque commodité sur

chaque chemin est $((x_p^k)_{p\in P^k})_{k\in K}$. Dans un deuxième temps, un chemin est sélectionné pour chaque commodité. La probabilité que le chemin p soit sélectionné dans P^k est x_p^k. Chaque commodité est ensuite affectée au chemin sélectionné pour créer une solution insécable. Cette procédure produit, avec une probabilité arbitrairement élevée, une solution insécable dont la congestion est $O\left(\frac{\ln|E|}{\ln\ln|E|}\right)$ plus grande que celle de la solution fractionnaire. L'algorithme de Raghavan et Tompson (1987) est donc un algorithme d'approximation de facteur $O\left(\frac{\ln|E|}{\ln\ln|E|}\right)$.

Par la suite, nous proposons une amélioration de cet algorithme qui permet d'obtenir de meilleurs résultats en pratique. Par ailleurs, nous donnons en Section 3.3 une preuve complète pour un facteur d'approximation qui, en fonction de la taille des commodités, est meilleur que celui proposé par Raghavan et Tompson (1987).

3.2 L'algorithme d'arrondi aléatoire séquentiel SRR

Nous présentons maintenant l'algorithme *Sequential Randomized Rounding* (SRR) qui est une heuristique polynomiale gloutonne pour le problème de flot insécable. Cet algorithme est une extension de l'algorithme de Raghavan et Tompson (1987) similaire à celui proposé par Coudert et Rivano (2002) pour le problème de *Light Path Assignment*. Nous ajoutons cependant quelques subtilités spécifiques à notre problème comme le tri des commodités par ordre décroissant de demande. L'algorithme SRR donne également la possibilité de calculer moins souvent la relaxation linéaire du problème pour réduire le temps de calcul global de l'algorithme. Par rapport à l'algorithme d'approximation de Raghavan et Tompson (1987), l'algorithme SRR a un temps d'exécution plus long, aucune garantie de performance mais fournit des solutions de meilleure qualité sur les instances testées. En tant qu'heuristique, l'algorithme SRR a un temps de calcul plus court que les méthodes de résolutions exactes et les métaheuristiques, en particulier pour les grandes instances.

3.2.1 Présentation de l'algorithme

L'algorithme SRR (Algorithme 1) alterne entre deux étapes différentes : résoudre la relaxation linéaire du problème et fixer certaines commodités à un chemin unique. Pour le problème de flot insécable, la relaxation linéaire est un problème de flot multi-commodités. Par souci de clarté nous utilisons les notations de la formulation par chemins dans ce qui suit. En revanche, dans les expérimentations, une formulation arc-nœud couplée à un solveur commercial (Gurobi Optimization, 2020) et au regroupement de commodités présenté en Section 3.2.3 est utilisée pour résoudre la relaxation linéaire. Des solveurs spécialisés plus efficaces ou des algorithmes d'approximation pour le problème de flot multi-commodités sont présents dans la littérature (voir Section 2.5). La résolution de la relaxation linéaire fournit une distribution du flot entre les chemins possibles pour chaque commodité : $((x_p^k)_{p\in P^k})_{k\in K}$. Un chemin est ensuite sélectionné parmi les chemins possibles pour certaines commodités. Ces commodités seront obligées d'utiliser le chemin qui leur est affecté pour le reste de l'algorithme. Ce chemin est choisi en utilisant la procédure d'arrondi aléatoire introduite par Raghavan et Tompson

(1987) : pour la commodité k, le chemin p est sélectionné avec la probabilité x_p^k. La probabilité que la commodité k utilise l'arc e est donc $f_e^k = \sum_{p \in P^k | e \in p} x_p^k$. Lors de la résolution des prochaines relaxations linéaires, les commodités fixées seront également forcées de n'utiliser que leur chemin affecté.

La principale différence entre l'heuristique SRR et l'algorithme RR de Raghavan et Tompson (1987) est que, dans SRR, la relaxation linéaire est actualisée plusieurs fois au cours du processus d'arrondi aléatoire. Plus précisément, après avoir décidé de fixer certaines commodités à un unique chemin, la relaxation linéaire est à nouveau résolue en imposant que les commodités fixées n'utilisent que leur chemin affecté. Pour décider quand la relaxation linéaire est actualisée, la procédure suivante est utilisée. Dans la solution de la relaxation linéaire, certaines commodités utilisent plusieurs chemins. Une fois que θ de ces commodités sont fixées à un seul chemin, la relaxation linéaire est actualisée. Le choix du seuil θ est un compromis entre le temps de calcul et la qualité de la solution. Si la relaxation linéaire est souvent actualisée (seuil bas), les décisions de fixation prennent en compte la plupart des décisions de fixation précédentes mais cela demande un temps de calcul plus élevé. Si la relaxation linéaire est peu actualisée, les décisions de fixation ne prennent pas en compte les décisions de fixation précédentes mais le temps de calcul est faible. Une analyse de sensibilité de ce paramètre, donnée en Section 3.4, nous a mené à fixer la valeur du seuil θ à $\frac{|V|}{4}$ dans nos expériences.

Une autre différence avec l'algorithme RR est que les solutions sont créées en utilisant la fonction objectif de somme des dépassements de capacité présentée en Section 2.1.1 au lieu de la fonction objectif de congestion classique. Comme expliqué en Section 3.5.1, lors de l'utilisation de la fonction objectif de somme des dépassements de capacité, l'heuristique SRR renvoie des solutions ayant un dépassement de capacité plus faible mais aussi une congestion plus faible.

Par rapport à l'algorithme proposé par Coudert et Rivano (2002), l'heuristique SRR offre plus de flexibilité sur le nombre d'actualisations de la relaxation linéaire grâce au paramètre θ. De plus, la principale différence est que le problème de flot insécable présent dans (Coudert et Rivano, 2002) découle d'un problème de *Light Path Assignment*. Cela implique que les demandes de toutes les commodités sont unitaires et donc que les chemins des commodités sont choisis sans ordre particulier. Or l'heuristique SRR tire parti du fait que, dans le cas général des flots insécables, les commodités possédent des demandes différentes. Ainsi, les chemins sont affectés aux commodités par ordre décroissant de demande des commodités, c'est-à-dire les commodités de plus grande demande sont affectées en premier. Cet ordre est classiquement utilisé dans les heuristiques des problèmes de *Bin Packing* tel que l'heuristique *Next Fit Decreasing* (Csirik *et al.*, 1986). La logique derrière cet ordre est d'affecter d'abord les commodités ayant une grande demande, tant qu'il reste beaucoup de capacité les arcs. Les commodités dont la demande est moindre sont ensuite utilisées pour combler les vides restants. En Section 3.4, les résultats expérimentaux montrent que cet ordre d'arrondi des variables a un impact important sur la qualité des solutions fournies par l'heuristique.

Algorithme 1 Heuristique SRR

Entrées : $G = (V, E, c)$ un graphe avec des capacités sur ces arcs, $L = (O^k, D^k, d^k)_{k \in K}$ une liste de commodités, θ un seuil

1: Trier les commodités par ordre décroissant de demande
2: $K_{fixed} = \varnothing$ ▷ K_{fixed} contient les commodités fixées à un seul chemin
3: Poser $C = \theta$.
4: **Pour** chaque commodité k^* par ordre décroissant de demande **faire**
5: **Si** $C \geq \theta$ **alors**
6: Poser $C = 0$.
7: $(P^k, (x_p^k)_{p \in P^k})_{k \in K} =$ Résoudre_Relaxation_Linéaire $\left(G, L, K_{fixed}, (p_k)_{k \in K_{fixed}}\right)$
8: Choisir un chemin p^* dans P^{k^*} avec probabilité $x_{p^*}^{k^*}$
9: Ajouter l'index k^* à K_{fixed}.
10: $p^{k^*} = p^*$
11: **Si** $x_{p^*}^{k^*} < 1$ **alors**
12: Poser $C = C + 1$
13: **Renvoyer** $(p^k)_{k \in K}$

3.2.2 Analyse de complexité

Au cours de chacune ces $|K|$ itérations, l'algorithme effectue les actions suivantes :

— Résolution de la relaxation linéaire : requiert $O(LR(|V|, |E|, |K|))$ opérations où $LR(|V|, |E|, |K|)$ est la complexité de la résolution de la relaxation linéaire (Ligne 7).

— Sélection du chemin p^* : comme le flot de chaque commodité peut être décomposé en au plus $|E|$ chemins, au plus $|E|$ variables x_p^k prennent une valeur non nulles (Ford Jr, 1956). Choisir l'une de ces variables requiert donc $O(|E|)$ opérations (Ligne 8).

De plus, trier les commodités requiert $O(|K| \log(|K|))$ opérations (Ligne 1). La complexité totale est donc $O(|K|(\log |K| + |E| + LR(|V|, |E|, |K|)))$.

3.2.3 Groupement des commodités par origine

Afin de résoudre la relaxation linéaire du problème, nous utilisons une technique de regroupement de commodités par origine. Le regroupement des commodités par origine consiste à considérer, dans la relaxation linéaire, un ensemble de commodités de même origine comme une seule commodité. Cette technique est reliée aux techniques d'agrégation de variables présente dans la littérature des flots multi-commodités sécable (Bauguion *et al.*, 2013, 2015). La nouvelle commodité possède une demande égale à la somme des demandes initiales. Pour s'assurer que le flot aille vers les bonnes destinations, un nœud de super-destination est créé et connecté à chaque destination initiale avec un arc de capacité égale à la demande initiale de la commodité. Lors de la résolution de la relaxation linéaire, la solution renvoyée est la même avec ou sans groupement des commodités à condition que toutes les commodités d'un

groupe partent du même nœud. Le regroupement de commodités réduit considérablement le temps de calcul d'une solution linéaire calculée à l'aide d'un solveur de programmation linéaire lorsque le nombre d'origines différentes est beaucoup plus petit que le nombre de commodités. Dans nos instances, inspirées de problèmes de télécommunications, les commodités sont émises à partir d'un petit nombre d'origines différentes, il est donc efficace de regrouper les commodités par origine. Cependant, les solutions produites lorsque les commodités sont regroupées ne renvoient pas toute l'information nécessaire à la poursuite de l'algorithme. Il est nécessaire de calculer exactement les chemins utilisés par chaque commodité. Un algorithme de décomposition de flot effectue ceci rapidement en $O(|V|(|E| + |K|))$ opérations (Ford Jr, 1956).

3.3 Analyse du facteur d'approximation d'une variante de SRR

Dans cette section, nous présentons l'algorithme *Constrained Sequential Randomized Rounding* (CSRR). Cette variante de l'heuristique SRR possède des garanties d'approximation similaires à celles de l'algorithme RR de Raghavan et Tompson (1987). Les résultats d'approximation sont obtenus en considérant la fonction objectif de congestion classique. En effet, pour l'objectif de somme des dépassements de capacité, la valeur de la solution optimale peut être nulle. Cela se produit lorsque le flot de toutes les commodités respecte les capacités des arcs. Dans ce cas, tout algorithme d'approximation doit trouver la solution optimale. Ainsi, la fonction objectif de somme des dépassements de capacité n'est pas adaptée aux preuves d'approximation à facteur multiplicatif.

L'algorithme RR renvoie une solution de congestion moins de $O\left(\frac{\ln(|E|\epsilon^{-1})}{\ln(\ln(|E|\epsilon^{-1}))}\right)$ fois plus grande que la solution optimale avec probabilité $1-\epsilon$. Nous étendons et renforçons l'analyse des algorithmes d'arrondi aléatoire en donnant un nouveau facteur d'approximation égale à $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ pour les algorithmes RR et CSRR. Dans ce nouveau facteur, γ est un paramètre de granularité de l'instance égal à $\frac{d_{max}}{c_{min}\Delta^*}$ où Δ^* est la congestion optimale dans la relaxation linéaire. Ce paramètre est petit lorsque les commodités sont petites par rapport à $c_{min}\Delta^*$. Le paramètre γ peut être relié au paramètre $\frac{d_{max}}{c_{min}}$ introduit par Chakrabarti *et al.* (2007); Shepherd et Vetta (2015); Azar et Regev (2006) pour renforcer leur analyse d'approximation dans le cas de l'objectif de demande servie maximale. Le nombre γ est un paramètre décisif des instances de flot insécable. En effet, il reste constant lorsque les capacités et les demandes sont uniformément multipliées par une constante.

Pour prouver un facteur d'approximation pour un algorithme d'arrondi aléatoire où la relaxation linéaire est actualisée (comme à la ligne 7 de l'Algorithme 1), il apparait nécessaire d'ajouter une contrainte dans la relaxation linéaire. Ainsi, l'algorithme CSRR est identique à l'algorithme SRR à l'exception de la contrainte supplémentaire suivante dans la relaxation linéaire :

$$\sum_{k \in K \setminus K_{fixed}} f_e^k d^k \leq c_e \Delta^* - \sum_{k \in K_{fixed}} \hat{f}_e^k d^k, \ \forall e \in E, \tag{3.1}$$

où Δ^* est la congestion optimale de la première relaxation linéaire ; K_{fixed} est l'ensemble des commodités qui ont été fixées à un unique chemin avant la résolution de relaxation linéaire actuelle et f_e^k sont les variables utilisées pour optimiser le flot des commodités non fixées. Les commodités dans K_{fixed} envoyaient leur flot sur plusieurs chemins dans la relaxation linéaire avant d'être fixées sur un chemin ; $\hat{f}_e^k$ est le flot fractionnaire correspondant sur chaque arc pour ces commodités. Notons que pour le calcul de la première relaxation linéaire, cette contrainte n'a aucun impact sur la solution et peut être supprimée. Ainsi, cette contrainte disparaît dans l'algorithme de Raghavan et Tompson (1987), puisqu'il n'y a qu'une seule résolution de la relaxation linéaire.

Par souci de clarté, nous construisons la preuve suivante dans le cas où l'algorithme CSRR ne ferait qu'une seule actualisation de la solution de relaxation linéaire, juste après la première étape d'arrondi. L'extension au cas de plusieurs actualisations effectuées à tout moment peut se faire par récurrence.

Dans ce qui suit, soient F_e^k les variables aléatoires discrètes indiquant le flot de la commodité k sur l'arc e dans la solution fournie par l'algorithme CSRR. Les variables F_e^k prennent la valeur d^k avec probabilité $f_e^k = \sum_{p \in P^k | e \in p} x_p^k$ et 0 sinon. Ainsi, leur espérance $\mathbb{E}[F_e^k] = f_e^k d^k = d^k \sum_{p \in P^k | e \in p} x_p^k$ est aussi le flot de la commodité k sur l'arc e dans la solution de la relaxation linéaire. Soit k_1 l'indice de la commodité de plus grande demande (donc la première à être affectée dans l'Algorithme 1) et soit $F_e = \sum_{k \in K \setminus \{k_1\}} F_e^k$. Une fois conditionnées par $F_e^{k_1}$, les variables aléatoires F_e^k $(k \neq k_1)$ sont indépendantes les unes des autres car la relaxation linéaire n'est pas actualisée entre leurs étapes d'arrondi respectives. Cependant, elles ne sont pas indépendantes de $F_e^{k_1}$; en particulier, on a $\mathbb{E}[F_e^k | F_e^{k_1}] \neq \mathbb{E}[F_e^k]$. En effet la réalisation de $F_e^{k_1}$ dans la première étape d'arrondi conditionne la résolution de la relaxation linéaire ultérieure. Ainsi, elle conditionne les valeurs f_e^k qui paramètrent la distribution des variables aléatoires F_e^k. Pour rappeler cette dépendance, nous notons $f_e^k(F_e^{k_1})$ le flot fractionnaire de la commodité k sur l'arc e. Notons que la contrainte (3.1) ajoutée dans l'algorithme CSRR peut être réécrite en termes de variables aléatoires :

$$\mathbb{E}[F_e | F_e^{k_1}] \leq c_e \Delta^* - \mathbb{E}[F_e^{k_1}].$$

Rappelons que dans la formulation de la congestion, la fonction objectif vise à minimiser le facteur minimum par lequel il faut multiplier les capacités de tous les arcs afin de pouvoir affecter un chemin à chaque commodité. Nous notons C^* la congestion optimale de l'instance de flots insécables considérée. Comme présenté ci-dessus, $F_e^{k_1} + F_e$ est le flot sur arc e dans la solution fournie par l'algorithme CSRR. Ainsi, prouver un facteur d'approximation probabiliste $(1 + \alpha)$ pour cet algorithme revient à prouver que pour tous les arcs, $F_e^{k_1} + F_e$ reste inférieur à $(1 + \alpha) c_e C^*$ avec une forte probabilité. Formellement, on cherche à prouver que pour un ϵ petit :

$$\mathbb{P}\left(\forall e \in E, F_e^{k_1} + F_e \leq (1 + \alpha) c_e C^*\right) \geq 1 - \epsilon.$$

A l'inverse, cela revient à prouver que, avec probabilité au plus ϵ, il existe un arc e où la congestion dépasse $(1 + \alpha) C^*$:

$$\mathbb{P}\left(\exists e \in E, F_e^{k_1} + F_e \geq (1 + \alpha) c_e C^*\right) \leq \epsilon.$$

Pour cela, nous prouvons dans le Théorème 3.1 que pour tout arc e :

$$\mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \leq \frac{\epsilon}{|E|}.$$

En effet, dans ce cas, comme Δ^* borne inférieurement C^*, on a :

$$\begin{aligned}
\mathbb{P}\left(\exists e \in E, F_e^{k_1} + F_e \geq (1+\alpha)c_eC^*\right) &= \mathbb{P}\left(\bigvee_{e\in E} F_e^{k_1} + F_e \geq (1+\alpha)c_eC^*\right) \\
&\leq \mathbb{P}\left(\bigvee_{e\in E} F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \\
&\leq \sum_{e\in E} \mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \\
&\leq \sum_{e\in E} \frac{\epsilon}{|E|} \\
&= \epsilon
\end{aligned}$$

Pour nous assurer que $\mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \leq \frac{\epsilon}{|E|}$, nous prouvons d'abord dans le Lemme 3.2 que la probabilité $\mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right)$ est bornée par une quantité $g_e(\alpha)$. Le Lemme 3.2 est prouvé en utilisant l'inégalité de Markov et la traduction probabiliste de la contrainte (3.1). Prouver le Lemme 3.2 nécessite un résultat préliminaire introduit dans le Lemme 3.1. Enfin, la preuve est achevée en montrant qu'il existe une valeur de α satisfaisant $1+\alpha = O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ et pour tout arc $g_e(\alpha) \leq \frac{\epsilon}{|E|}$

Sans perte de généralité et pour supprimer d_{max} de la preuve, les instances sont supposées normalisées de telle sorte que $d_{max} = 1$ et donc $\gamma = (c_{min}\Delta^*)^{-1}$. Nous présentons maintenant les deux lemmes avec leur preuve.

Lemme 3.1
Pour tout scalaire positif α, $\mathbb{E}\left[(1+\alpha)^{F_e}|F_e^{k_1}\right] \leq e^{\alpha\mathbb{E}[F_e|F_e^{k_1}]}$.

Démonstration.

$$\begin{aligned}
\mathbb{E}\left[(1+\alpha)^{F_e}|F_e^{k_1}\right] &= \mathbb{E}\left[\prod_{k\in K\setminus\{k_1\}}(1+\alpha)^{F_e^k}|F_e^{k_1}\right] \\
&= \prod_{k\in K\setminus\{k_1\}}\mathbb{E}\left[(1+\alpha)^{F_e^k}|F_e^{k_1}\right] \quad \text{car les } F_e^k|F_e^{k_1} \text{ sont indépendante} \\
&= \prod_{k\in K\setminus\{k_1\}}(f_e^k(F_e^{k_1})(1+\alpha)^{d^k}+1-f_e^k(F_e^{k_1})) \\
&\leq \prod_{k\in K\setminus\{k_1\}}(f_e^k(F_e^{k_1})(1+\alpha d^k)+1-f_e^k(F_e^{k_1})) \quad \text{car } d^k\leq 1 \\
&= \prod_{k\in K\setminus\{k_1\}}(1+\alpha f_e^k(F_e^{k_1})d^k) \\
&\leq \prod_{k\in K\setminus\{k_1\}}e^{\alpha f_e^k(F_e^{k_1})d^k} \\
&= e^{\alpha\sum_{k\in K\setminus\{k_1\}}f_e^k(F_e^{k_1})d^k} \\
&= e^{\alpha\mathbb{E}[F_e|F_e^{k_1}]}
\end{aligned}$$

□

Lemme 3.2
Pour tout scalaire positif α et toute instance du problème de flot insécable, le flot $F_e^{k_1}+F_e$ renvoyé par l'algorithme CSRR sur l'arc e satisfait :

$$\mathbb{P}(F_e^{k_1}+F_e\geq(1+\alpha)c_e\Delta^*)\leq\left[\frac{e^\alpha}{(1+\alpha)^{1+\alpha}}\right]^{c_e\Delta^*}$$

Démonstration. On note $\delta=(1+\alpha)^{(1+\alpha)c_e\Delta^*}$

$$\begin{aligned}
&\mathbb{P}(F_e^{k_1}+F_e\geq(1+\alpha)c_e\Delta^*) \\
&= \mathbb{P}\left((1+\alpha)^{F_e^{k_1}+F_e}\geq\delta\right) \\
&\leq \delta^{-1}\mathbb{E}\left[(1+\alpha)^{F_e^{k_1}+F_e}\right] && \text{d'après l'inégalité de Markov} \\
&= \delta^{-1}\mathbb{E}\left[\mathbb{E}[(1+\alpha)^{F_e^{k_1}+F_e}|F_e^{k_1}]\right] \\
&= \delta^{-1}\mathbb{E}\left[(1+\alpha)^{F_e^{k_1}}\mathbb{E}[(1+\alpha)^{F_e}|F_e^{k_1}]\right] \\
&\leq \delta^{-1}\mathbb{E}\left[(1+\alpha)^{F_e^{k_1}}e^{\alpha\mathbb{E}[F_e|F_e^{k_1}]}\right] && \text{d'après le Lemme 1}
\end{aligned}$$

$$
\begin{aligned}
&\leq \delta^{-1}\mathbb{E}\left[(1+\alpha)^{F_e^{k_1}} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}])}\right] && \text{d'après la contrainte (3.1)}\\
&= \delta^{-1} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}])}\mathbb{E}\left[(1+\alpha)^{F_e^{k_1}}\right]\\
&= \delta^{-1} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}])}(f_e^{k_1}(1+\alpha)^{D_{k_1}} + 1 - f_e^{k_1})\\
&\leq \delta^{-1} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}])}(f_e^{k_1}(1+\alpha D_{k_1}) + 1 - f_e^{k_1}) && \text{car } D_{k_1} \leq d_{max} \leq 1\\
&\leq \delta^{-1} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}])} e^{\alpha f_e^{k_1} D_{k_1}}\\
&= \delta^{-1} e^{\alpha(c_e\Delta^* - \mathbb{E}[F_e^{k_1}] + \mathbb{E}[F_e^{k_1}])}\\
&= \delta^{-1} e^{\alpha c_e \Delta^*}\\
&= \left[\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right]^{c_e\Delta^*}
\end{aligned}
$$

□

Nous présentons maintenant le théorème principal qui borne pour chaque arc la probabilité que la solution renvoyée par l'algorithme CSRR ait une congestion élevée.

Théorème 3.1
Pour tout $\epsilon > 0$, il existe un facteur d'approximation $1+\alpha$ en $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ tel que, pour toute instance du problème de flot insécable, le flot $F_e^{k_1} + F_e$ renvoyé par l'algorithme CSRR sur l'arc e satisfasse :

$$\mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \leq \frac{\epsilon}{|E|}.$$

Démonstration. Le Lemme 3.2 donne $\mathbb{P}(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*) \leq \left[\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right]^{c_e\Delta^*}$. Pour garantir la véracité du théorème, nous devons montrer qu'il existe un scalaire $1+\alpha$ qui est $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ et pour chaque arc satisfait :

$$\left[\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right]^{c_e\Delta^*} \leq \frac{\epsilon}{|E|}$$

$$\Longleftrightarrow$$

$$(1+\alpha)\ln(1+\alpha) - \alpha \geq \frac{\ln(|E|\epsilon^{-1})}{c_e\Delta^*}$$

Pour l'arc de capacité c_{min} qui donne la borne la plus élevée, la borne inférieure est $B = \gamma \ln(|E|\epsilon^{-1})$ (rappelons que $d_{max} = 1$). Ainsi nous étudions la solution de l'équation $(1+\alpha)\ln(1+\alpha) - \alpha = B$ et montrons qu'elle satisfait $1+\alpha = O\left(\frac{B}{\ln(B)}\right)$. En remplaçant

$\ln(1+x)$ par les bornes classiques $2\frac{x-1}{x+1} \leq \ln(1+x) \leq x$, nous obtenons :

$$\begin{aligned} & \alpha^2 \geq (1+\alpha)\ln(1+\alpha) - \alpha \geq \alpha - 2 \\ \iff \quad & \alpha^2 \geq B \geq \alpha - 2 \\ \iff \quad & \sqrt{B} \leq \alpha \leq B + 2 \end{aligned}$$

ce qui implique :

$$B = (1+\alpha)\ln(1+\alpha) - \alpha \geq (1+\alpha)\ln(1+\sqrt{B}) - B - 2$$

et finalement,

$$1 + \alpha \leq \frac{2B+2}{\ln(1+\sqrt{B})} \sim \frac{4B}{\ln(B)} = O\left(\frac{B}{\ln(B)}\right)$$

Ainsi, la solution de l'équation $(1+\alpha)\ln(1+\alpha) - \alpha = \gamma \ln(|E|\epsilon^{-1})$ satisfait $1 + \alpha = O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ et pour cette valeur de α, nous avons

$$\mathbb{P}\left(F_e^{k_1} + F_e \geq (1+\alpha)c_e\Delta^*\right) \leq \frac{\epsilon}{|E|}.$$

□

En utilisant le Théorème 3.1, nous avons montré que, pour toute instance du problème de flot insécable, l'algorithme CSRR renvoie une solution dont la congestion est inférieure à $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\Delta^*\right)$ avec probabilité $1-\epsilon$. Pour l'algorithme RR de Raghavan et Tompson (1987), le Lemme 3.2 et le Théorème 3.1 s'appliquent toujours et la démonstration du Lemme 3.2 est plus simple car la commodité k_1 n'a pas besoin d'être traitée séparément. Les résultats d'approximation sont donc identiques pour l'algorithme RR.

Lors des résolutions de la relaxation linéaire dans l'algorithme CSRR, la contrainte (3.1) est très restrictive. Elle ne laisse presque aucune flexibilité aux variables pour s'éloigner de l'optimum précédent. Nous illustrons maintenant cette propriété avec un exemple.

Exemple : Nous appelons ici première relaxation linéaire le programme linéaire résolu avant qu'aucune commodité ne soit fixée à un seul chemin et seconde relaxation linéaire le programme linéaire résolu après que le premier ensemble de commodités soit fixé à un seul chemin. Pour la première et la seconde relaxation linéaire respectivement, on note f_e^1 et f_e^2 le flot total sur l'arc e de toutes les commodités qui ne sont pas fixées lors de la première étape d'arrondi. Supposons que la première relaxation linéaire ait une solution optimale unique dans laquelle chaque arc a la même congestion Δ^*. Dans la seconde relaxation linéaire, la contrainte (3.1) *impose que, $\forall e \in E, f_e^1 \geq f_e^2$. Dans ce cas, cela implique que $\forall e \in E, f_e^1 = f_e^2$. Sinon, le flot des commodités non fixées dans la première relaxation linéaire pourrait être remplacé par le même flot provenant de la seconde relaxation linéaire, créant ainsi une nouvelle solution optimale de la première relaxation linéaire. Cela contredirait l'hypothèse selon laquelle la première relaxation linéaire a une solution optimale unique. Ainsi, dans cet exemple, les*

variables de la deuxième relaxation linéaire ne peuvent pas du tout s'éloigner de l'optimum précédent.

Cet exemple met en évidence que la deuxième relaxation linéaire ne peut s'éloigner de la solution de la première relaxation linéaire qu'en utilisant le jeu laissé entre la congestion de chaque arc et Δ^*. Ainsi, lorsque la contrainte (3.1) est utilisée, l'étape d'actualisation n'a qu'un faible impact sur la solution de la relaxation linéaire et l'algorithme CSRR ne bénéficie pas beaucoup de l'étape d'actualisation. Nous conjecturons que c'est pourquoi l'algorithme CSRR ne donne pas de bons résultats expérimentaux par rapport à l'heuristique SRR (voir la Section 3.4). Pour surmonter ce problème, nous remplaçons la contrainte (3.1) par la contrainte suivante :

$$\sum_{k \in K_{free}} f_e^k d^k \leq \beta c_e \Delta^* - \sum_{k \in K_{fixed}} \hat{f}_e^k d^k, \ \forall e \in E. \tag{3.2}$$

En remplaçant Δ^* par $\beta\Delta^*$ et γ par $\beta^{-1}\gamma$ pour un $\beta \geq 1$ dans la preuve, il est possible de prouver qu'en utilisant la contrainte (3.2), le facteur d'approximation de l'algorithme CSRR devient $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\beta^{-1}\gamma \ln(|E|\epsilon^{-1}))}\right)$ ce qui reste $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$ pour tout β fixé. Contrairement à la contrainte (3.1), dans la plupart des cas pratiques, la contrainte (3.2) n'est pas active dans les solutions optimales des relaxations linéaires même pour $\beta = 1.1$. Ainsi, la contrainte (3.2) n'a aucun impact sur les calculs pratiques et permet à l'algorithme CSRR de donner les mêmes résultats expérimentaux que l'heuristique SRR.

Pour résumer, dans cette section, nous avons ajouté une contrainte à l'heuristique SRR et utilisé la formulation utilisant la congestion classique comme fonction objectif afin de créer un algorithme ayant les mêmes garanties d'approximation que l'algorithme RR de Raghavan et Tompson (1987). Nous avons également renforcé l'analyse d'approximation des deux algorithmes en introduisant un paramètre de granularité γ. Ce paramètre a la propriété de rester constant lorsque les demandes des commodités et les capacités des arcs sont uniformément multipliées par une constante. Enfin, nous avons légèrement modifié la contrainte ajoutée pour atténuer son impact dans les calculs pratiques. Cette modification augmente le facteur d'approximation d'une valeur négligeable.

3.4 Étude expérimentale

Dans cette section, nous présentons des expériences soutenant nos affirmations : l'heuristique SRR a un temps de calcul plus court sur les grandes instances que les méthodes exactes et donne des solutions ayant un meilleur dépassement de capacité que l'algorithme de Raghavan et Tompson (1987) et les métaheuristiques de la littérature. L'impact du tri des commodités par ordre décroissant de demande dans les algorithmes d'arrondi aléatoire est également étudié. De plus, l'heuristique SRR est comparée à l'algorithme d'approximation CSRR. Les jeux de données et le code utilisés dans cette section expérimentale sont accessibles à l'url `https://github.com/SuReLI/randomized_rounding_paper_code`. Le code a été écrit en Python 3 et les expériences ont été faites sur un serveur avec 48 CPU Intel Xeon

E5-2670 2.30 GHz, 60 Gbit de RAM et CentOS Linux 7.

Dans cette section, chaque figure présente les résultats pour un jeu d'instances. Chaque jeu d'instances comprend dix groupes d'une centaine d'instances, chaque groupe contenant des instances créées à l'aide des mêmes paramètres. Une exception est faite dans la Figure 3.2 dans laquelle chaque groupe ne contient que dix instances. Chaque point d'une figure rapporte, pour un algorithme, le résultat moyen d'un groupe d'instances. Les intervalles de confiance à 95% sont également représentés sous forme de boîtes semi-transparentes autour de la courbe principale.

3.4.1 Jeux d'instances

Comme indiqué par Masri *et al.* (2019), aucun jeu d'instances standard n'est présent dans la littérature pour le problème de flot insécable, en particulier pour de grandes instances. En effet, la plupart des travaux utilisent de petits graphes (moins de 50 nœuds) sur lesquels ils génèrent à leur façon un ensemble de commodités. Les plus grandes instances (100 nœuds) peuvent être trouvées dans les travaux de Masri *et al.* (2015) et Li *et al.* (2010). Masri *et al.* (2015) utilisent un graphe grille tandis que Li *et al.* (2010) créent des graphes avec une version adaptée du générateur de graphes NETGEN. Pour compenser cette absence d'instances standards, nous présentons de manière détaillée notre procédure de génération d'instances. De plus, toutes nos instances sont données dans notre GitHub avec le code utilisé pour leur génération.

Dans nos tests, nous considérons deux types de graphes : des graphes aléatoires fortement connexes et des graphes grille. Pour les graphes aléatoires fortement connexes, nous utilisons la méthode suivante pour construire un graphe fortement connexe aléatoire très peu dense : sélectionner un nœud aléatoire u, sélectionner un nœud v aléatoire tel qu'il n'y ait pas de chemin de u à v, ajouter un arc (u, v) au graphe, répéter jusqu'à ce que le graphe soit fortement connexe. Des arêtes aléatoires sont ensuite ajoutées pour contrôler le degré moyen du graphe (toujours supérieur à 2). Dans nos tests, le degré moyen est fixé à 5 et la probabilité qu'un nœud soit une origine est de 1/10. Un graphe grille est une grille torique $n \times m$ avec p nœuds supplémentaires. Chaque nœud supplémentaire est connecté aléatoirement à q nœuds sur la grille et sert d'origine du flot des commodités. Ce schéma émule un constellation de satellite telle que présentée au Chapitre 1 Dans nos tests, nous utilisons $m = n = p = \frac{q}{2}$. Sauf indication contraire, les capacités des arcs sont de 10^4. Ces choix reflètent les données issues de la constellation Telesat.

Pour les deux types de graphes, les commodités sont créées une à une jusqu'à ce qu'aucune commodité ne puisse plus être créée sans violer les contraintes de capacité. Une commodité O, D, d est créée comme suit :

— choisir un nœud destination D ;

— choisir une origine O qui peut accéder à D en respectant les capacités restantes ;

— calculer un chemin simple aléatoire p de O à D en utilisant une recherche en profondeur où l'ordre de visite des nœuds nouvellement découverts est aléatoire ;

— choisir une demande d ; pour ce choix, le paramètre $\hat{d}_{max}$ définit la demande maximale possible d'une commodité et $U(x)$ est une variable entière aléatoire uniformément tirée dans $[1, x]$; deux méthodes sont envisagées pour choisir la demande ; soit $d = \min(c_p, U(\hat{d}_{max}))$ soit $d = U(\min(c_p, \hat{d}_{max}))$; la deuxième méthode crée de nombreuses petites commodités ce qui rend les instances plus faciles à résoudre, cette méthode n'est pas utilisée dans ce chapitre ; sauf indication contraire $\hat{d}_{max} = 1500$;

— diminuer les capacités sur le chemin p de d ;

— ajouter (O, D, d) à la liste des commodités créées.

Notons que les demandes créées de cette manière peuvent toujours être routées en respectant les capacités des arcs et que la congestion optimale est donc unitaire. Ainsi, nous savons que les solutions optimales ont un dépassement de capacité nul et une congestion unitaire. Par conséquent, ces valeurs optimales n'ont pas besoin d'être calculées avec des algorithmes exacts. Notons également que le fait d'avoir des commodités rentrant dans les capacités du graphe tout en les remplissant complètement a tendance à créer des instances difficiles à résoudre. En effet, s'il y a trop peu de commodités, la plupart des solutions n'ont aucun dépassement de capacité et l'instance est facile à résoudre. D'un autre côté, s'il y a beaucoup trop de demandes pour tenir dans les capacités, une création de dépassement de capacité est inévitable, indépendemment du chemin choisi pour chaque commodité et, encore une fois, l'instance est facile à résoudre.

3.4.2 Comparaison des métaheuristiques de la littérature

Dans cette section, nous présentons une comparaison de deux métaheuristiques de la litérature introduite en Section 2.2.3 : ACO-MC, l'un des algorithmes d'optimisation par colonies de fourmis présenté par Li *et al.* (2010), et la recherche à voisinage variable (VNS) de Masri *et al.* (2015). Les deux algorithmes ont été reproduits et le code utilisé est donné dans notre dépôt GitHub. Nous comparons ces algorithmes avec un recuit simulé de notre cru. Le but est de n'utiliser que la meilleure métaheuristique comme comparaison pour l'algorithme SRR dans les sections suivantes. L'algorithme de Li *et al.* (2010) a été codé exactement comme présenté dans leur article.

Implémentation de (Masri et al.*, 2015) :* en raison de différences dans le problème de flot insécable considéré, la partie de recherche locale de leur algorithme a dû être modifiée. Leur recherche locale crée un nouveau chemin pour une commodité en partant de sa destination et en ajoutant des arcs au chemin jusqu'à ce que l'origine soit atteinte. A chaque étape, l'arc suivant est choisi en considérant les informations heuristiques suivantes pour chaque arc sortant du dernier nœud du chemin courant :

$$I_e = \frac{1}{l_e} + \left(1 - \frac{1}{\hat{c}_e}\right)$$

où l_e est le délai de l'arc e (*i.e.* sa longueur) et $\hat{c}_e$ est la capacité restante sur l'arc e. Comme nous n'avons pas de délai sur les arcs dans notre problème, l_e a été fixé à 1 pour chaque arc. De plus, dans le problème étudié par Masri *et al.* (2015), la capacité restante $\hat{c}_e$ est

positive car aucun dépassement de capacité n'est autorisé. Parce que nous pouvons avoir des capacités restantes négatives, nous remplaçons la fonction $f : x \to 1 - \frac{1}{x}$ appliquée à $\hat{c}_e$ par $g : x \to \frac{1}{2}\left(1 + \frac{x}{1+|x|}\right)$. La fonction g a été choisie pour avoir des propriétés similaires à la fonction f.

$$\forall x \in (1, +\infty),\ 0 < f(x) < 1$$

$$\forall x \in (-\infty, +\infty),\ 0 < g(x) < 1$$

$$f(x) - 1 \underset{+\infty}{\sim} g(x) - 1 \underset{+\infty}{\sim} \frac{-1}{x}$$

$$g(x) \underset{-\infty}{\sim} \frac{-1}{x}$$

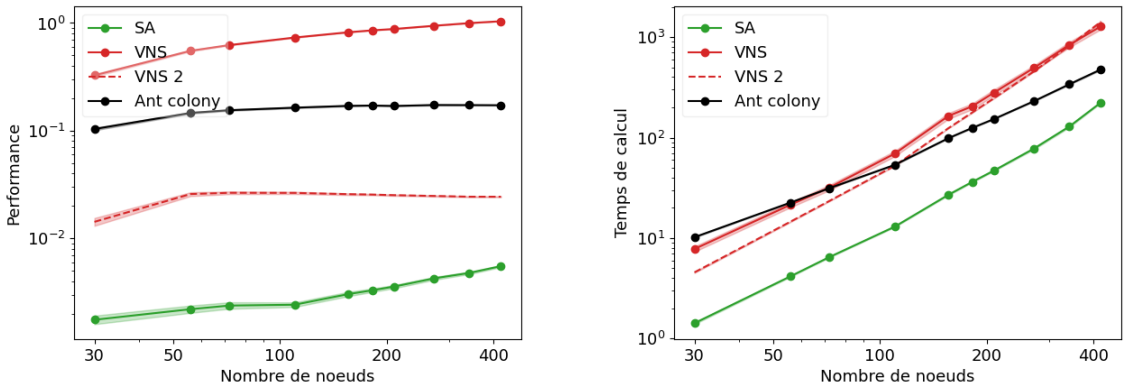
Dans nos tests, nous présentons également les résultats obtenus avec une version de l'algorithme de Masri *et al.* (2015) où la recherche locale est désactivée.

Notre recuit simulé : à chaque itération, une solution est créée au voisinage de la solution courante ; cette modification est acceptée avec une probabilité dépendant de l'amélioration/détérioration de la solution et d'un paramètre de température. A chaque itération, le paramètre de température est multiplié par $1 - \epsilon$ pour un certain petit ϵ fonction du nombre d'itérations. Au début de la procédure de recuit simulé et de manière similaire à l'algorithme de Masri *et al.* (2015), une liste de k-plus-courts chemins de longueur 10 est calculée pour chaque commodité en utilisant l'algorithme de Jiménez et Marzal (1999). Pour initialiser la solution, chaque commodité prend un chemin aléatoire dans sa liste de k-plus-courts chemins. A chaque itération, une solution voisine est créée en remplaçant le chemin d'une commodité par un chemin choisi au hasard dans sa liste des k-plus-courts chemins. Le critère d'arrêt est le nombre total d'itérations.

Valeur des hyper-paramètres : nous avons comparé différentes valeurs d'hyper-paramètres pour ACO-MC mais avons finalement décidé d'utiliser les mêmes valeurs que dans (Li *et al.*, 2010). À l'exception de la taille du plus grand voisinage qui n'est pas mentionnée, les deux versions de la recherche à voisinage variable utilisent les valeurs d'hyper-paramètres données par Masri *et al.* (2015). Après avoir testé différentes valeurs, la taille du plus grand voisinage a été fixée à 3. Les hyper-paramètres du recuit simulé sont la température initiale et la température finale choisies respectivement à 200 et 1. Le recuit simulé (SA) effectue $2|K|^{1.5}$ itérations pour avoir un temps de calcul comparable à l'algorithme SRR dans les sections suivantes. ACO-MC et le VNS de Masri *et al.* (2015) effectuent respectivement 50 et 100 itérations. Bien que ces nombres semblent très faibles, il faut considérer que pour chacune de leurs itérations, les algorithmes de Li *et al.* (2010) et Masri *et al.* (2015) génèrent respectivement $|K|$ et $\frac{|K|}{2}$ nouveaux chemins. Avec ces nombres d'itérations, ils nécessitent déjà un temps de calcul plus long que la procédure de recuit simulé. Enfin, la variation de la recherche de voisinage variable où la recherche locale est désactivée (VNS 2) effectue $|K|^{1.5}$ itérations.

Résultats : La Figure 3.1 présente les résultats de chaque algorithme sur un jeu d'instances composé de graphes grille de différentes tailles. La procédure de recuit simulé surpasse clairement les autres algorithmes sur ce jeu de données. Nous avons obtenu des résultats similaires sur tous les autres jeux de données. C'est pourquoi l'algorithme de recuit simulé a été choisi

dans les sections suivantes comme point de comparaison pour l'algorithme SRR.



(a) Dépassement de capacité divisé par la demande totale — graphes grille

(b) Temps de calcul en secondes — graphes grille

FIGURE 3.1 – Performance et temps de calcul en fonction du nombre de nœuds pour différentes métaheuristiques

Discussion : nous essayons maintenant d'expliquer pourquoi les métaheuristiques de la littérature sont surpassées par notre recuit simulé. À chaque itération, la procédure de colonie de fourmis passe la plupart de son temps de calcul à créer une solution entièrement nouvelle. En effet, un nouveau chemin est créé pour chaque commodité pour un total de $|K|$ chemins générés. À l'inverse, à chaque itération, la procédure de recuit simulé ne choisit qu'un seul chemin dans une liste (choisir un chemin est beaucoup plus rapide que d'en créer un). Cela permet à la procédure de recuit simulé d'avoir un nombre d'itérations beaucoup plus grand et d'évaluer beaucoup plus de solutions. Quant à l'algorithme de Masri *et al.* (2015), il semble que la recherche locale fonctionne mal dans nos tests. Ceci est en partie dû au fait que la procédure de génération de chemin ne sait pas où se trouve son nœud cible jusqu'à ce qu'elle l'ait rencontré. En effet, les informations heuristiques utilisées pour choisir l'arc suivant du chemin prennent en compte la longueur l_e des arcs et non la longueur du chemin le plus court vers le nœud cible. Ainsi, lors du choix de l'arc suivant, la procédure ne sait pas si elle se rapproche ou s'éloigne de son objectif. Nous avons essayé de remplacer l_e par la longueur du chemin le plus court vers le nœud cible dans la recherche locale, mais désactiver complètement la procédure donne de meilleurs résultats. Enfin, il semble que la recherche à voisinage variable ne soit pas un bon choix de métaheuristique pour notre version du problème de flot insécable (surtout lorsque le nombre de commodités est supérieur à mille). En effet, une recherche à voisinage variable utilise plusieurs voisinages de la taille varie entre 1 et un valeur maximale que nous notons N. Dans le cas de l'algorithme de Masri *et al.* (2015), deux solutions sont voisines pour le j-iéme voisinage si moins de j commodités y utilisent des chemins différents. En faisant varier la taille N du plus grand voisinage, il est apparu que la meilleure valeur pour ce paramètre est la valeur 1 ce qui est le cas où il n'y a qu'un seul voisinage.

3.4.3 Résultats expérimentaux

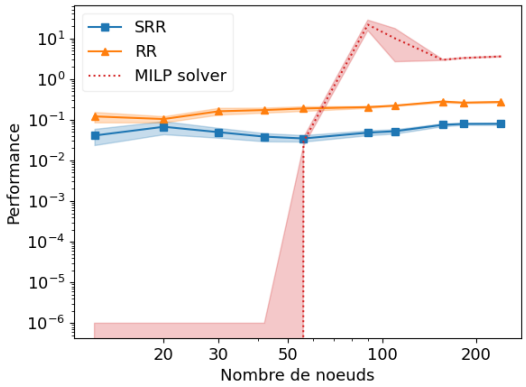
Nous présentons maintenant les résultats expérimentaux principaux de ce chapitre. Les algorithmes suivants sont comparés dans nos tests :

— RR : l'algorithme d'arrondi aléatoire de Raghavan et Tompson (1987) ;
— RR trié : une version de l'algorithme RR où les commodités sont arrondies par ordre décroissant de demande ;
— SRR : l'heuristique d'arrondi aléatoire séquentiel décrite en Section 3.2.1 ;
— SRR unsorted : une version de l'heuristique SRR où les commodités ne sont pas arrondies par ordre décroissant de demande mais dans un ordre aléatoire ;
— CSRR : la version de l'algorithme SRR ayant des propriétés d'approximation, présentée en Section 3.3 ;
— SA et SA2 : notre procédure de recuit simulée présentée en Section 3.4.2 ; SA effectue $2|K|^{1.5}$ itérations pour avoir un temps de calcul similaire à SRR sur les graphes grille tandis que SA2 effectue $6|K|^{1.5}$ itérations ;
— Solveur MILP : formulation arc-nœud résolue avec Gurobi 8.11.

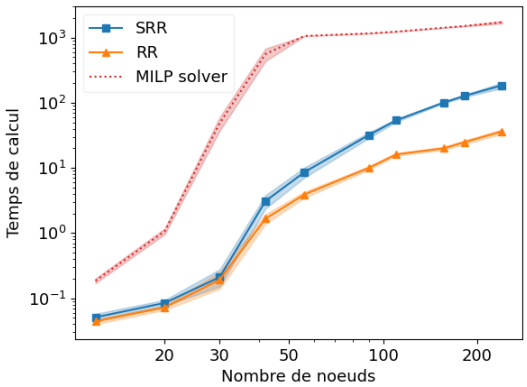
Tous les algorithmes d'arrondi aléatoire de cette liste utilisent l'objectif de somme des dépassements de capacité dans la relaxation linéaire pour créer leurs solutions.

Une caractéristique importante de l'heuristique SRR est sa capacité à traiter de grandes instances. Ainsi, nous illustrons d'abord comment l'algorithme surpasse une méthode MILP dans ce contexte. Dans la Figure 3.2, une méthode MILP basée sur la formulation arc-noeud est comparée aux algorithmes RR et SRR sur des petits graphes grilles avec peu de commodités. Pour ces tests, la capacité des arcs est de 3 et la demande maximale d'une commodité est de 2 pour maintenir faible le nombre de commodités et de variables de la formulation MILP. L'algorithme MILP résout de manière optimale toutes les petites instances mais donne de mauvais résultats sur les grandes instances dans un délai de 20 minutes. En comparaison, les autres méthodes conservent des performances raisonnables sur les grandes instances. En raison de la grande quantité de mémoire requise, l'algorithme MILP n'a pas pu être testé sur les autres ensembles de données plus volumineux.

La Figure 3.3 compare différents algorithmes d'arrondi aléatoire et l'algorithme de recuit simulé sur des graphes grille et des graphes aléatoires fortement connexes. Dans les deux cas, l'algorithme SRR nécessite un temps de calcul plus long que l'arrondi aléatoire pur mais renvoie des solutions de bien meilleure qualité. Pour les petites instances de graphes grille, les meilleurs résultats sont donnés par la procédure de recuit simulé. Cependant, à mesure que le nombre de nœuds augmente, l'heuristique SRR surpasse la procédure de recuit simulé avec le même temps de calcul (SA). De plus, sur les plus grandes instances, la procédure de recuit simulé est dépassée par l'heuristique SRR même si elle est autorisée à utiliser trois fois plus de temps de calcul (SA2). Quant aux graphes aléatoires fortement connectés, avec les paramètres décrits, l'heuristique SRR renvoie les solutions de la plus haute qualité mais nécessite un temps



(a) Dépassement de capacité divisé par la demande totale — graphes grille



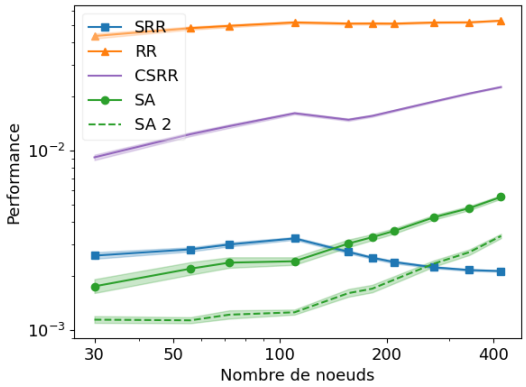
(b) Temps de calcul en secondes — graphes grille

FIGURE 3.2 – Performance et temps de calcul en fonction du nombre de nœuds pour de petites instances

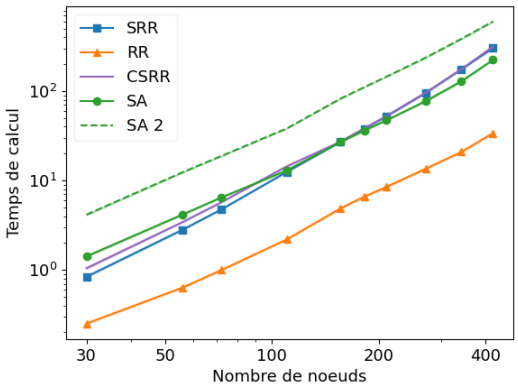
de calcul plus long que le recuit simulé. Il semble que la résolution de la relaxation linéaire prenne plus de temps sur les graphes aléatoires que sur les graphes grille. L'algorithme CSRR renvoie des solutions moins bonnes que l'heuristique SRR dans un temps de calcul similaire. Comme expliqué à la fin de la Section 3.3, ceci est dû à la contrainte (3.1) ajoutée dans la résolution de la relaxation linéaire. Il est possible d'assouplir cette contrainte en la remplaçant par la contrainte (3.2) tout en gardant un facteur d'approximation similaire. En pratique, la contrainte (3.2) n'est jamais active dans la solution optimale de la relaxation linéaire. Ainsi, avec cette adaptation, l'algorithme CSRR renvoie exactement les mêmes solutions que l'heuristique SRR.

La Figure 3.4 décrit la qualité de solution et le temps de calcul pour des graphes grille avec un nombre constant de nœuds et d'arcs mais un nombre croissant de commodités. Cette croissance est obtenue en modifiant le rapport entre la capacité des arcs et la demande maximale des commodités. Le Tableau 3.1 présente les paramètres utilisés pour créer les instances. Comme on peut le voir sur la Figure 3.4a, tous les algorithmes produisent des solutions de meilleure qualité lorsqu'il y a un grand nombre de commodités (c'est-à-dire lorsque les commodités ont une faible demande par rapport aux capacités des arcs). Comme expliqué en Section 3.3, pour les algorithmes d'arrondi aléatoire, ce résultat pratique est lié à la présence du paramètre de granularité γ dans le facteur d'approximation des algorithmes RR et CSRR. La Figure 3.4b montre que les algorithmes d'arrondi aléatoire ont un temps de calcul qui évolue très bien avec le nombre de commodités. Cela peut être dû à notre choix d'une formulation arc-nœud agrégée pour résoudre la relaxation linéaire. En revanche, le recuit simulé nécessite une forte augmentation de son nombre d'itérations pour obtenir les résultats rapportés sur la Figure 3.4a.

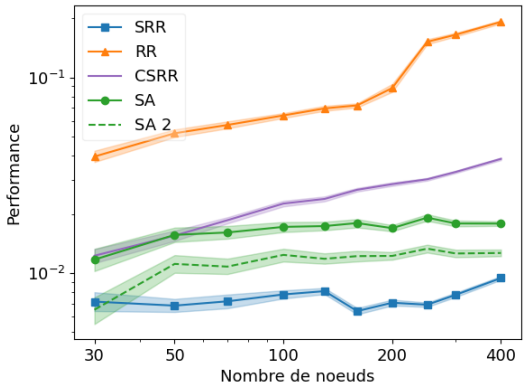
Nous analysons maintenant l'impact de l'arrondi des commodités par ordre décroissant de demande, sur la base des résultats présentés dans la Figure 3.5. L'arrondi aléatoire pur



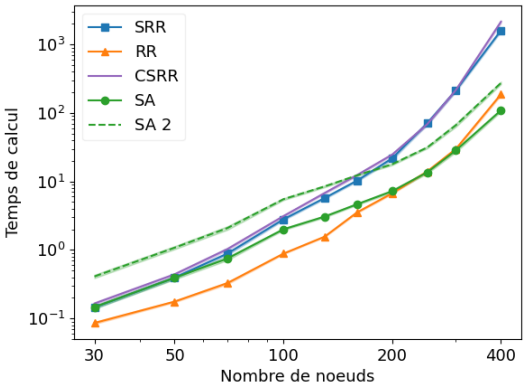
(a) Dépassement de capacité divisé par la demande totale — graphes grille



(b) Temps de calcul en secondes — graphes grille



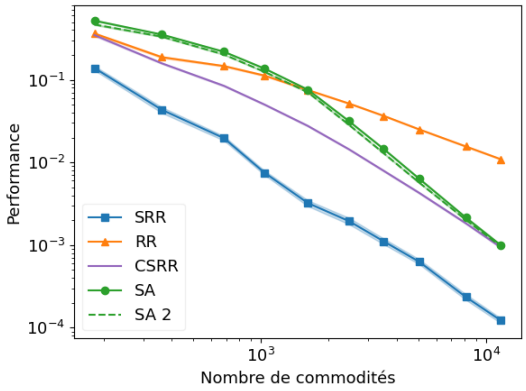
(c) Dépassement de capacité divisé par la demande totale — graphes aléatoires



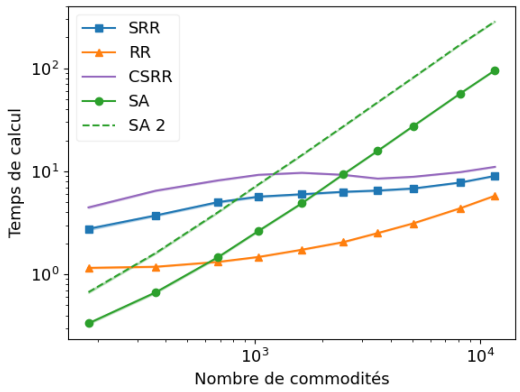
(d) Temps de calcul en secondes — graphes aléatoires

FIGURE 3.3 – Performance et temps de calcul en fonction du nombre de nœuds pour de grandes instances

et l'heuristique SRR donnent des solutions de meilleure qualité lorsque les commodités sont triées mais l'impact est significativement plus élevé sur l'algorithme SRR. Une explication possible d'un impact aussi fort est donnée en Section 3.5.2. À première vue, un ordre d'arrondi ne devrait pas avoir d'impact sur l'algorithme de Raghavan et Tompson (1987) puisque les commodités sont arrondies indépendamment. Cependant, les résultats indiquent clairement le contraire et cela est dû à la façon dont la relaxation linéaire est calculée dans nos expériences. En effet, en utilisant la formulation arc-nœud de la Section 2.1.2 et l'agrégation des commodités présentée en Section 3.2.3, le programme linéaire ne donne pas directement une distribution du flot pour chaque commodité. La distribution de flot renvoyée concerne les commodités agrégées. Pour obtenir le flot de chaque commodité, un algorithme de décomposition de flot est utilisé. Cet algorithme a tendance à moins diviser les commodités qui sont décomposées en premier. Ainsi, lorsque les commodités de plus grande demande sont arrondies en premier, elles sont également décomposées en premier. Cela signifie que les plus



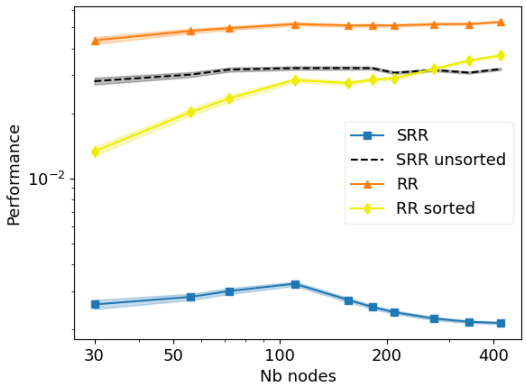
(a) Dépassement de capacité divisé par la demande totale — graphes grille



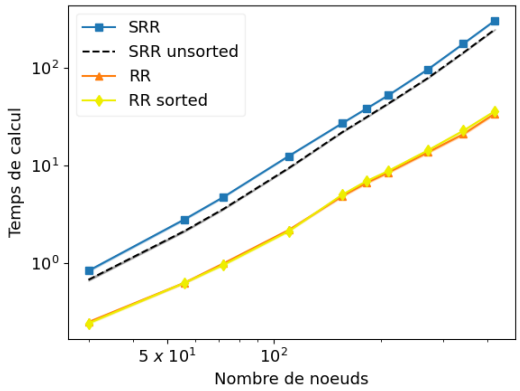
(b) Temps de calcul en secondes — graphes grille

FIGURE 3.4 – Performance et temps de calcul en fonction du nombre de commodités

grosses commodités ont moins de chance d'être scindées et donc moins de chance que leur arrondi crée un important dépassement de capacité.



(a) Dépassement de capacité divisé par la demande totale — graphes grille



(b) Temps de calcul en secondes — graphes grille

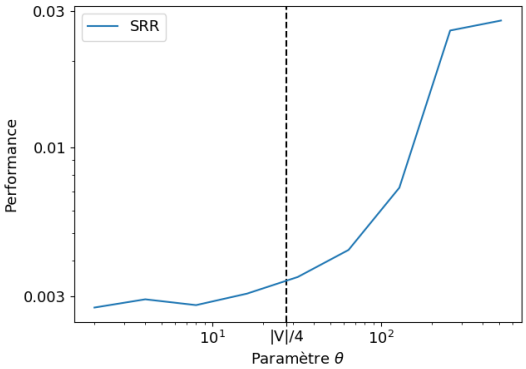
FIGURE 3.5 – Influence du tri des commodités sur la performance et le temps de calcul

Enfin, nous décrivons comment nous avons choisi la valeur $\frac{|V|}{4}$ pour le paramètre d'actualisation θ de l'algorithme SRR présenté en Section 3.2.1. La définition de cette valeur implique un compromis entre la qualité de la solution et le temps de calcul. La Figure 3.6 compare les performances de SRR pour différentes valeurs de ce paramètre, sur un graphe grille de 110 nœuds. Le temps de calcul de l'heuristique SRR diminue affinement avec le paramètre θ. Cependant, le dépassement de capacité de la solution renvoyée diminue de moins en moins lorsque θ diminue. Nous avons décidé de choisir θ de telle sorte que la solution renvoyée soit proche de la meilleure solution possible tout en économisant beaucoup de temps de calcul par rapport au choix de θ proche de zéro. En répétant cette expérience sur des graphes de tailles

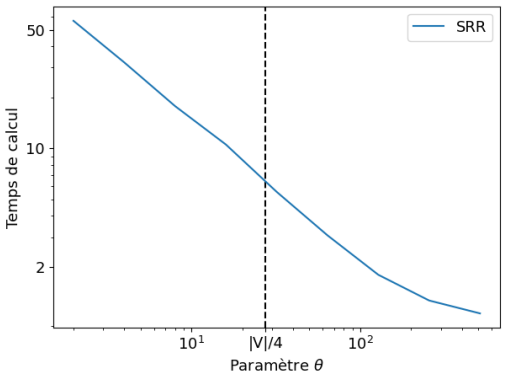
Capacité des arcs	1	2	5	10	20
Demande maximale des commodités	1	1	2	3	4
Nombre moyenne de commodités	182	362	685	1038	1615
Capacité des arcs	50	100	200	500	1000
Demande maximale des commodités	7	10	14	22	31
Nombre moyenne de commodités	2462	3512	5048	8138	11644

TABLE 3.1 – Paramètre des instances du jeu de données où le nombre de commodités est variable : les graphes ont 110 nœuds et 580 arcs

et de types différents, il est apparu que la valeur de compromis choisie était proche de $\frac{|V|}{4}$. Cette valeur est représentée par une ligne pointillée dans la Figure 3.6.



(a) Dépassement de capacité divisé par la demande totale — graphes grille



(b) Temps de calcul en secondes — graphes grille

FIGURE 3.6 – Performance et temps de calcul en fonction la valeur du paramètre θ de SRR

3.5 Discussion sur SRR

Dans cette section, nous discutons différentes propriétés des algorithmes d'arrondi aléatoire. Nous montrons d'abord que l'utilisation de la fonction objectif de somme des dépassements de capacité présentée en Section 2.1.1 au lieu de la fonction objectif de congestion classique a un impact positif sur les résultats pratiques. Ensuite, nous mettons en évidence une interaction positive entre le tri des commodités et l'actualisation de la relaxation linéaire.

3.5.1 Impact de la fonction objectif

Deux types de fonction objectif coexistent dans l'heuristique SRR. Le premier est la fonction objectif du problème de flot insécable qui sera appelé *métrique d'évaluation* dans cette section. La seconde est la fonction objectif utilisée dans les résolutions de la relaxation linéaire

	Dépassement de capacité	Congestion	Mixte
Moyenne	1.025	1.049	1.027
Écart-type	0.016	0.019	0.012

TABLE 3.2 – Comparaison de la congestion des solutions renvoyées par l'heuristique SRR en utilisant différents objectifs de génération

à l'intérieur de l'heuristique SRR que nous appellerons *objectif de génération*. Pour les deux types précédents, nous étudions les deux fonctions présentées en Section 2.1.1 : la somme du dépassement de capacité et la congestion. En général, l'objectif de génération et la métrique d'évaluation sont choisis identiques. Cependant, la somme des dépassements de capacité et la congestion sont des métriques très proches et nous voulons étudier l'impact de l'utilisation de l'un pour créer des solutions pour l'autre. Nous montrons ci-dessous que l'utilisation de la somme des dépassements de capacité au lieu de la congestion comme objectif de génération donne des solutions de meilleure qualité pour les deux métriques d'évaluation.

Lorsque la métrique d'évaluation est la somme des dépassements de capacité, dans nos tests, les solutions générées en utilisant la somme des dépassements de capacité ont une qualité de solution dix fois supérieure à celles créées à l'aide de la congestion. De façon plus surprenante, la somme des dépassements de capacité utilisée comme objectif de génération donne également de meilleurs résultats lorsque la métrique d'évaluation est la congestion. Pour tester cela, l'heuristique SRR a été appliquée avec les deux objectifs de génération sur 100 instances ayant des graphes grille de 120 nœuds, 700 arcs et 4500 commodités. La congestion moyenne des solutions renvoyées est indiquée dans les deux premières colonnes du Tableau 3.2 avec l'écart-type des résultats. Les congestions moyennes pour les deux objectifs de génération sont séparées par plus d'un écart type en faveur de la somme des dépassements de capacité. En utilisant les valeurs données dans les deux premières colonnes du Tableau 3.2, un test de Student à deux échantillons non appariés a été réalisé. Le test a montré que la somme des dépassements de capacité utilisée comme objectif de génération fonctionne significativement mieux que la congestion. La p-valeur associée au test est de 10^{-18}.

Nous supposons que la différence observée ci-dessus entre les deux objectifs de génération s'explique par le raisonnement suivant. Dans la relaxation linéaire, la fonction objectif de congestion ne différencie pas les solutions ayant un arc fortement congestionné des solutions à plusieurs arcs fortement congestionnés. Cependant, un plus grand nombre d'arcs congestionnés dans la relaxation linéaire implique, en moyenne, une plus grande congestion dans la solution renvoyée par le processus d'arrondi aléatoire. De plus, si le processus d'arrondi aléatoire crée des arcs avec une congestion plus grande que celle de la relaxation linéaire, alors, dans les actualisations ultérieures, les contraintes de capacité sont levées pour tous les autres arcs. Cela conduit à des solutions insécables encore plus congestionnées.

Avec cette conjecture à l'esprit, nous avons créé un algorithme qui utilise la congestion comme objectif principal de génération mais fonctionne aussi bien que l'algorithme utilisant la somme des dépassements de capacité. Pour cela, un bi-objectif est fixé dans la relaxation linéaire : trouver la solution de somme des dépassements de capacité minimale parmi les solu-

tions de congestion minimale. Les résultats obtenus avec cet objectif de génération alternatif sont présentés dans la troisième colonne du Tableau 3.2 sous le nom d'objectif " Mixte ". Ces résultats sont comparables à ceux obtenus avec l'objectif de somme des dépassements de capacité. En effet, un test de Student à deux échantillons non appariés effectué sur les valeurs données dans les première et troisième colonnes du Tableau 3.2 ne montre pas de différence statistiquement significative. La p-valeur associée au test est de 0.32.

3.5.2 Interaction entre le tri et l'actualisation

Dans cette section, nous discutons de l'interaction positive observée en Section 3.4 entre le tri des commodités par ordre décroissant de demande et l'actualisation de la relaxation linéaire. Pour cela, nous étudions un exemple jouet où le graphe est composé de deux nœuds reliés par deux arcs parallèles. Nous supposons également que la somme des demandes est égale à la somme des capacités. Nous comparons les solutions avec l'objectif de somme des dépassements de capacité et supposons qu'aucune solution binaire n'a un dépassement de capacité nul.

Lors du calcul de la relaxation linéaire, les extrema optimaux du polyèdre sont tels que chaque commodité est affectée à un arc sauf une qui est scindée entre les deux arcs. Nous supposons que toutes les commodités ont la même probabilité d'être la commodité scindée. Après avoir arrondi aléatoirement une solution linéaire de ce type, la solution binaire obtenue a un dépassement de capacité compris entre zéro et la demande de la commodité scindée.

Si la relaxation linéaire est actualisée après que la commodité scindée soit fixée, la solution obtenue devrait en moyenne être meilleure. En effet, après actualisation de la solution linéaire, trois situations peuvent se produire. Premièrement, les commodités libres restantes ne peuvent pas changer d'arc pour compenser le dépassement de capacité généré en arrondissant la commodité scindée. Dans ce cas, aucune commodité ne change son flot et la nouvelle solution linéaire est binaire et cette solution est la même avec ou sans actualisation. Deuxièmement, les commodités libres restantes ne compensent qu'une partie du dépassement de capacité créé. Dans ce cas, même si certaines commodités changent d'arc, la nouvelle solution linéaire est une solution binaire dont le dépassement de capacité est inférieur à celui avant de la solution présente avant l'étape d'actualisation. Dans le dernier cas, les commodités libres restantes compensent complètement le dépassement de capacité créé et une nouvelle commodité scindée est créée. L'application d'un arrondi aléatoire à partir de cette solution linéaire donne une solution dont le dépassement de capacité est compris entre zéro et la demande de la nouvelle commodité scindée. Étant donné que les commodités sont fixées à un seul chemin par ordre décroissant de demande, la nouvelle commodité scindée a une demande plus faible que l'ancienne commodité scindée. La plage dans laquelle le dépassement de capacité de la solution finale peut varier est donc plus petite ce qui devrait en moyenne conduire à une meilleure solution. En outre, de nouvelles commodités scindées sont créées jusqu'à ce que les commodités libres ne puissent plus compenser complètement l'arrondi de la dernière commodité scindée. La plupart du temps, cela se produit lorsqu'il ne reste que quelques commodités et donc de petites commodités. Dans ce cas, la dernière commodité scindée a une

faible demande et l'algorithme produit un petit dépassement de capacité final.

Finalement, nous pouvons voir que dans cet exemple jouet, actualiser la solution linéaire et trier les commodités par demandes décroissantes donne, en moyenne, des solutions avec un dépassement de capacité plus petit que l'arrondi aléatoire pur. Nous conjecturons que ce phénomène se généralise à n'importe quel graphe. En effet, le fait que les commodités divisées deviennent de plus en plus petites au fur et à mesure que l'algorithme progresse devrait conduire à des dépassements de capacité finaux de plus en plus petits.

3.6 Conclusion

Dans ce chapitre, nous avons présenté une heuristique basée sur l'arrondi aléatoire pour les instances de grande taille du problème de flot insécable qui étend l'algorithme de Raghavan et Tompson (1987). Nous avons montré expérimentalement que sur des instances de grande taille, cette heuristique produit des solutions ayant un plus petit dépassement de capacité que les autres méthodes avec laquelle elle a été comparée. En particulier, cette heuristique semble prometteuse dans le cadre de la gestion du débit de la constellation Telesat. De plus, en restreignant l'actualisation de la relaxation linéaire, nous avons également créé une variante de l'heuristique ayant des propriétés d'approximation. Le facteur d'approximation de cette variante et de l'algorithme de Raghavan et Tompson (1987) a ensuite été amélioré jusqu'à $O\left(\frac{\gamma \ln(|E|\epsilon^{-1})}{\ln(\gamma \ln(|E|\epsilon^{-1}))}\right)$. Ce nouveau facteur d'approximation dépend d'un paramètre de granularité γ et permet de comprendre le comportement de l'arrondi aléatoire lorsque les demandes des commodités sont petites par rapport aux capacités (*i.e.* $\gamma \ll 1$). De plus, le comportement de l'heuristique présentée a été analysé pour mettre en évidence deux de ses particularités clés. Premièrement, la nouvelle fonction objectif utilisée dans l'algorithme donne des solutions de meilleure qualité que celles créées en utilisant la fonction objectif de congestion classique. Deuxièmement, l'actualisation de la solution de la relaxation linéaire améliore les performances de l'heuristique lorsque les commodités sont triées par ordre de demande décroissante.

Même si les techniques discutées dans cet article ont été présentées dans le contexte des flots insécables, elles s'appliquent à d'autres contextes où la méthode d'arrondi aléatoire de Raghavan et Tompson (1987) est utilisée (problèmes de conditionnement, de recouvrement...). Leurs performances et leurs variations dans ces contextes pourraient être étudiées. De plus, une direction de recherche prometteuse pourrait être de considérer une remise en cause de certaines des décisions prises lors des algorithmes d'arrondi aléatoire.

L'une des limitations du problème de flot insécable statique est qu'il ne prend pas en compte la dynamique de la constellation, car il ne la considère qu'à un instant fixé. C'est pourquoi nous étudions dans le prochain chapitre le problème de flot insécable dynamique qui permet de prendre en compte cet aspect du problème dans la modélisation. Comme dans ce chapitre, nous y étudierons des algorithmes capables de résoudre des instances de moyenne et grande taille. C'est pourquoi nous réutiliserons l'heuristique introduite dans ce chapitre dans les méthodes de résolution proposées.

Chapitre 4

Le problème de flot insécable dynamique

Résumé du chapitre

Dans ce chapitre, nous nous intéressons au problème de flot insécable dynamique. Ce problème permet de prendre en compte la dynamique de la constellation en plus des autres contraintes traitées dans le problème de flot insécable statique. Nous présentons plusieurs formulations modélisant le problème dynamique ou sa relaxation linéaire. De plus, nous présentons deux nouvelles approches pour résoudre le sous-problème rencontré dans la génération de colonnes utilisée pour résoudre la formulation par séquences de chemins proposée par Gamvros et Raghavan (2012). Ces méthodes ne possèdent pas les limitations de la méthode proposée par Gamvros et Raghavan (2012) et possèdent une meilleure complexité dans le pire cas. Nous présentons également différentes approches de résolution basées sur les formulations connues pour le problème de flot insécable dynamique. Ces algorithmes sont comparés sur des jeux de données comportant des instances de petite et moyenne taille. Plusieurs aspects clés de ces approches expliquant leurs performances sont mis en évidence à travers une étude par ablation.

Dans le chapitre précédent, nous nous sommes intéressés au problème de flot insécable. Ce problème modélise l'acheminement du flux des utilisateurs dans une constellation de satellites à un instant fixé. Dans ce chapitre, nous considérons une extension du problème de flot insécable permettant de prendre en compte la dynamique de la constellation que nous nommerons le problème de flot insécable dynamique. Dans ce problème, une séquence de problèmes de flot insécable est donnée, représentant différents pas de temps. Chaque pas de temps introduit quelques changements au problème : certaines commodités changent d'origine ou de destination, certains arcs sont ajoutés ou supprimés. Cette séquence de problèmes présente deux objectifs possiblement contradictoires : 1) minimiser le dépassement des capacités des arcs par le flot des commodités, 2) les commodités doivent, si possible, utiliser le même chemin pour des pas de temps consécutifs (c'est-à-dire minimiser le nombre de changements de chemin au fil du temps). Dans les applications industrielles, ce problème peut comporter un très grand nombre de pas de temps. Le problème peut alors être résolu en considérant un horizon glissant et uniquement quelques pas de temps à la fois. Pour rendre cette méthode

envisageable, nous considérons qu'une instance du problème de flot insécable dynamique est donnée avec un chemin initial pour chaque commodité qui représente le dernier chemin utilisé avant l'horizon considéré.

Nous commençons par présenter plusieurs formulations modélisant le problème de flot insécable dynamique ou sa relaxation linéaire. Puis, nous présentons deux nouvelles approches pour résoudre le sous-problème rencontré dans la génération de colonnes utilisée pour résoudre la formulation par séquences de chemins proposée par Gamvros et Raghavan (2012). Ces méthodes ne possèdent pas les limitations de la méthode proposée par Gamvros et Raghavan (2012) et possèdent une meilleure complexité dans le pire cas. Nous présentons ensuite différentes approches de résolution basés sur les formulations connues pour le problème de flot insécable dynamique. Ces algorithmes sont comparés sur des jeux de données comportant des instances de petite et moyenne taille. Plusieurs aspects clés de ces approches expliquant leurs performances sont mis en évidence à travers une étude par ablation.

4.1 Formulations pour le problème de flot insécable dynamique

Dans cette section, nous présentons une formulation linéaire en nombres entiers pour le problème de flot insécable dynamique ainsi qu'une formulation de programmation linéaire modélisant la relaxation linéaire de ce problème lorsqu'un seul pas de temps est considéré. Notons que même dans le cas d'un seul pas de temps, le problème est différent d'un problème statique car des chemins privilégiés (correspondant à ceux utilisés au pas de temps précèdent) sont fournis. La relaxation linéaire de la formulation linéaire en nombres entiers est montrée être aussi forte que la relaxation de la formulation par séquences de chemins de Gamvros et Raghavan (2012). Par ailleurs, la formulation linéaire ne peut pas être transformée en une formulation linéaire en nombres entiers car les commodités y sont modélisées de manière agrégée. Cependant, elle sera utilisée dans nos expériences en conjonction avec une heuristique pour créer des solutions entières du problème de flot insécable dynamique. Pour des raisons de simplicité de présentation, nous considérons que les pénalités associées aux changements de chemin sont unitaires pour toutes les commodités et tous les pas de temps. L'extension au cas général n'implique pas de changements majeurs dans les algorithmes et les formulations. De plus, il est possible dans le cas général de considérer une pénalisation linéaire par morceaux du dépassement de capacité. Cependant, nous considérons dans ce travail que le dépassement de capacité cumulé sur tous les pas de temps et toutes les commodités n'est pas pénalisé tant qu'il ne dépasse pas un montant total B pour un pas de temps et qu'après ce montant, il est fortement pénalisé. Par ailleurs, nous utiliseront les notations présentées en début de manuscrit et rappelées dans les sections suivantes.

4.1.1 Formulation par chemins étendue

Les formulations arc-nœud et par chemin pour le problème statique, présentées en Section 2.1, peuvent être converties en formulations pour le problème de flot insécable dynamique. Pour la formulation par chemins, cette extension est faite en considérant plusieurs pas de temps et en ajoutant de nouvelles variables n^k_{pt} et de nouvelles contraintes $\forall p, x^k_{pt} - x^k_{p,t-1} \leq n^k_{pt}$ pour enregistrer les pénalités payées. La formulation obtenue est présentée ci-dessous. Cette formulation utilise un nombre polynomial de variables et de contraintes en le nombre de commodités et de pas de temps mais pas en le nombre de nœuds et d'arcs en raison du nombre exponentiel de chemins dans un graphe. Les variables de cette formulation ont la signification suivante :

— x^k_{pt} décide si la commodité k utilise le chemin p au pas de temps t ;

— o_{et} représente le dépassement de capacité sur l'arc e au pas de temps t ;

— o_t représente la quantité de dépassement de capacité dépassant au pas de temps t le montant B de dépassement non pénalisé ;

— n^k_{pt} prend la valeur un si la commodité k utilise le chemin p au pas de temps t mais pas au pas de temps $t-1$.

On note P^k_t l'ensemble des chemins utilisables par la commodité k au pas de temps t. L'ensemble des pas de temps T est étendu pour inclure un pas de temps 0. L'ensemble P^k_0 de chemins utilisables au pas de temps zéro ne contient que le chemin emprunté par la commodité k avant la période à optimiser. Notons que les variables x^k_{pt} sont toujours considérées comme existantes pour les chemins valides comme invalides mais que x^k_{pt} est automatiquement égal à zéro lorsque le chemin p n'est pas valide pour la commodité k au pas de temps t.

$$\min_{x^k_{pt}, n^k_{pt}, o_{et}, o_t} \sum_{k \in K, t \in T} n^k_{pt} + \sum_{t \in T} o_t \tag{4.1a}$$

tel que

$$\sum_{p \in P^k_t} x^k_{pt} = 1 \qquad \forall k \in K \ \forall t \in T \tag{4.1b}$$

$$\sum_{k \in K} \sum_{p \in P^k_t | e \in p} x^k_{pt}\, d^k_t \leq c_{et}(1 + o_{et}) \qquad \forall e \in E_t \ \forall t \in T \tag{4.1c}$$

$$\sum_{e \in E_t} o_{et} \leq B + o_t \qquad \forall t \in T \tag{4.1d}$$

$$x^k_{pt} - x^k_{p,t-1} \leq n^k_{pt} \qquad \forall p \in P^k_t \ \forall k \in K \ \forall t \in T \setminus \{0\} \tag{4.1e}$$

$$x^k_{pt} \in \{0,1\},\ n^k_{pt} \in \mathbb{R}^+,\ o_{et} \in \mathbb{R}^+,\ o_t \in \mathbb{R}^+ \qquad \forall p \in \bigcup_k P^k_t,\ \forall k \in K,\ \forall e \in E_t,\ \forall t \in T \tag{4.1f}$$

L'équation (4.1b) garantit qu'exactement un chemin est choisi pour chaque commodité à chaque pas de temps. Les équations (4.1c) et (4.1d) sont les contraintes de capacité : elles assurent que o_{et} représente le dépassement de capacité sur l'arc e au pas de temps t et que

o_t représente la quantité de dépassement au delà du seuil B au pas de temps t. L'équation (4.1e) garantit que n^k_{pt} prend la valeur 1 lorsqu'un changement de chemin se produit pour la commodité k entre les pas de temps t et $t-1$. Le fait que $x^k_{pt} \in \{0,1\}$ garantit que le flot est insécable.

4.1.2 Équivalence de la formulation par séquences de chemins et de la formulation par chemins étendue

Les problèmes combinatoires peuvent souvent être décrits mathématiquement à l'aide de plusieurs modèles de programmation linéaire en nombres entiers. Ces modèles sont généralement comparés à l'aide de leur nombre de variables et de contraintes mais aussi à l'aide de la qualité de leur relaxation linéaire. En effet, plus une formulation possède une relaxation linéaire forte, meilleures sont les procédures de *Branch and Bound* appliquées a cette formulation, et plus la relaxation linéaire donne d'information sur le problème en nombres entiers. Dans cette section, nous montrons que la formulation par séquences de chemins présentée en Section 2.4 et la formulation par chemins étendue ont des relaxations linéaires de même qualité.

Proposition 4.1
Soit V_1 la valeur de la relaxation linéaire de la formulation par chemins étendue et V_2 la valeur de la relaxation linéaire de la formulation par séquences de chemins, alors $V_1 = V_2$.

Démonstration. 1) Montrons d'abord que $V_1 \leq V_2$.

L'espace des solutions de la relaxation linéaire des deux formulations est l'intersection du polyèdre R induit par les deux contraintes de capacité (4.1c) et (4.1d) et d'un autre polyèdre qui diffère entre les deux formulations. Pour la formulation par chemins étendue, ce polyèdre est $Q_1 = \{x^k_{pt} \in [0,1],\ n^k_{pt} \in \mathbb{R}^+ | \text{contraintes (4.1b) et (4.1e)}\}$. La formulation par séquences de chemins est obtenue en appliquant une décomposition de Dantzig-Wolfe aux contraintes (4.1b) et (4.1e) de la formulation par chemins étendue. Ainsi, le polyèdre associé à sa relaxation linéaire est l'enveloppe convexe Q_2 des points entiers appartenant à Q_1. Comme pour toute décomposition de Dantzig-Wolfe et toute enveloppe convexe, nous avons $Q_2 \subseteq Q_1$. Ainsi, l'espace des solutions de la relaxation de la formulation par séquences de chemins $R \cap Q_2$ est plus petit que celui de la relaxation de la formulation par chemins étendue $R \cap Q_1$. En conséquence, la relaxation linéaire de la formulation par séquences de chemins est au moins aussi forte que celle de la formulation par chemins étendue : $V_1 \leq V_2$.

2) On montre maintenant que $V_2 \leq V_1$.

Pour ce faire, considérons une affectation des variables x^k_{pt} de la formulation par chemins étendue représentant une distribution du flot dans le temps. On va montrer que quelle que soit cette distribution, elle peut être représentée à l'aide d'une affection des variables x^k_s de la formulation par séquences de chemins. Autrement dit, on va montrer que cette distribution peut être exprimée comme une combinaison convexe de séquences de chemins. De plus,

il est essentiel que ces deux représentations induisent le même nombre de changements de chemin. Ceci nous permettra de conclure que les solutions accessibles dans le polyèdre Q_1 sont également accessibles dans Q_2. Nous présentons maintenant un algorithme constructif qui calcule une combinaison convexe de séquences de chemins correspondant à une affectation des variables x^k_{pt}. Cette décomposition est effectuée séparément pour chaque commodité et nous ne considérons donc qu'une seule commodité dans la suite. De plus, la pénalité payée entre les pas de temps t et $t+1$ dépend des chemins choisis aux pas de temps t et $t+1$ mais pas des chemins choisis avant le pas de temps t. Cette indépendance des pas de temps nous permet de présenter l'algorithme pour un seul pas de temps.

Dans ce qui suit, appelons $\lambda(s)$ le coefficient d'une séquence de chemin s et $\lambda(\Pi) = \sum_{s\in\Pi}\lambda(s)$ la somme des coefficients d'un ensemble Π de séquences de chemins. De plus, appelons Π^k_{pt} l'ensemble des séquences de chemins construites par l'algorithme pour la commodité k et utilisant le chemin p au pas de temps t.

Schéma de l'algorithme. L'algorithme construit un ensemble de séquences de chemins ainsi que leurs coefficients dans la combinaison convexe. Les séquences de chemins sont construites de manière incrémentale. Elles sont d'abord toutes initialisées avec un chemin pour le premier pas de temps, puis toutes étendues avec un chemin pour le second pas de temps, etc. Cette construction possède deux objectifs sous-jacents. Premièrement, nous voulons que $\lambda(\Pi^k_{pt}) = x^k_{pt}$, c'est à dire que la somme des coefficients des séquences de chemins utilisant le chemin p au pas de temps t pour la commodité k soit égale à x^k_{pt}. Ceci garantit que la combinaison de séquences de chemins représente la même distribution de flot que les variables x^k_{pt}. Deuxièmement, nous voulons en priorité étendre les séquences de chemins avec le chemin par lequel elles se terminent déjà. Nous montrerons que cela garantit que les séquences de chemins utilisées induisent un nombre minimum de changements de chemin.

Nous donnons un exemple d'exécution de l'algorithme dans le tableau 4.1. Dans cet exemple, la distribution du flot représentée par les variables x^k_{pt} est donné dans le Tableau 4.1a. Dans les autres tableaux nous présentons l'exécution de l'algorithme. Dans ces tableaux, la colonne t contient les différentes séquences de chemins créées par l'algorithme au moment des extensions du pas de temps t. Ces différentes séquences de chemins sont indexées par la lettre s.

Initialisation de la combinaison convexe des séquences de chemins. Pour chaque chemin p autorisé au premier pas de temps, l'algorithme crée une séquence de chemins contenant uniquement le chemin p avec un coefficient égal à x^k_{p1} et l'ajoute à Π^k_{p1}.

Puis, pour chaque pas de temps suivant t, l'algorithme procède comme suit.

Extensions prioritaires. L'algorithme commence d'abord par étendre autant que possible les séquences de chemins avec le chemin par lequel elles se terminent déjà. Pour chaque chemin p autorisé au pas de temps t, si $x^k_{pt} \geq x^k_{p,t-1}$ alors toutes les séquences de chemins dans $\Pi^k_{p,t-1}$ sont étendues avec le chemin p et ajoutées à Π^k_{pt}. Dans ce cas, à ce stade, $\lambda(\Pi^k_{pt}) \leq x^k_{pt}$. Sinon, si $x^k_{pt} < x^k_{p,t-1}$, l'algorithme trouve un sous-ensemble Π de séquences de chemins inclus dans $\Pi^k_{p,t-1}$ et une séquence de chemins $s \subseteq \Pi^k_{p,t-1}$ tels que : $s \notin \Pi$ et $\lambda(\Pi) \leq x^k_{pt} \leq \lambda(\Pi) + \lambda(s)$.

p \ t	1	2	3
a	0.5	0.3	0.4
b	0.2	0	0
c	0	0.5	0.3
d	0.3	0.2	0.3

(a) Distribution du flot : valeur des variables x^k_{pt} pour une commodité

s \ t	1	2	3
1	a | 0.5		
2	b | 0.2		
3	d | 0.3		
4			
5			
6			
7			
8			

(b) Initialisation : une séquence contenant un unique chemin est créée pour chaque chemin utilisé au premier pas de temps

s \ t	1	2	3
1	a | 0.5	aa | 0.3	
2	b | 0.2		
3	d | 0.3	dd | 0.2	
4			
5			
6			
7			
8			

(c) Extensions prioritaires pour $t = 2$: les séquences sont étendues avec le chemin par lequel elles finissent déjà. En cas de besoin, les coefficients sont diminués pour que la somme des coefficients des séquences finissant par un chemin p ne dépasse pas x^k_{p2}

s \ t	1	2	3
1	a | 0.5	aa | 0.3	
2	b | 0.2		
3	d | 0.3	dd | 0.2	
4		ac | 0.2	
5		bc | 0.2	
6		dc | 0.1	
7			
8			

(d) Autres extensions pour $t = 2$: les séquences dont le coefficient à été diminué (ici toutes) sont étendues avec des chemins pour lesquels il reste du flot à allouer (ici uniquement le chemin c)

s \ t	1	2	3
1	a | 0.5	aa | 0.3	aaa | 0.3
2	b | 0.2		
3	d | 0.3	dd | 0.2	ddd | 0.2
4		ac | 0.2	acc | 0.2
5		bc | 0.2	bcc | 0.1
6		dc | 0.1	
7			
8			

(e) Extensions prioritaires pour $t = 3$: les séquences sont étendues avec le chemin par lequel elles finissent déjà. Notons en particulier que les coefficients de bcc et dcc sont diminués (jusqu'à 0 pour dcc) de telle sorte que $\lambda(acc) + \lambda(bcc) + \lambda(dcc) = x^k_{c3}$

s \ t	1	2	3
1	a | 0.5	aa | 0.3	aaa | 0.3
2	b | 0.2		
3	d | 0.3	dd | 0.2	ddd | 0.2
4		ac | 0.2	acc | 0.2
5		bc | 0.2	bcc | 0.1
6		dc | 0.1	
7			bca | 0.1
8			dcd | 0.1

(f) Autres extensions pour $t = 3$: les séquences dont le coefficient à été diminué (ici bc et dc) sont étendues avec des chemins où il reste du flot à allouer (ici les chemins a et d)

TABLE 4.1 – Exemple d'exécution de l'algorithme utilisé dans la preuve

Les séquences de chemins dans Π seront étendues avec le chemin p et ajoutées à Π^k_{pt}. Puis, la séquence de chemins s sera "divisée" en deux parties pour s'assurer que $\lambda(\Pi^k_{pt})$ soit égal à x^k_{pt} après que l'une des deux parties soit étendue avec le chemin p. Pour effectuer la division de la séquence de chemins s, l'algorithme la duplique d'abord en deux séquences de chemins s_1 et s_2 qui recevront de nouveaux coefficients. La première copie s_1 reçoit un coefficient $x^k_{pt} - \lambda(\Pi)$ avant d'être étendue avec le chemin p et ajoutée à Π^k_{pt}. La deuxième copie s_2 reçoit un coefficient $\lambda(s) - \lambda(s_1)$ et reste dans $\Pi^k_{p,t-1}$. Ce choix de nouveaux coefficients assure que $\lambda(s_1) + \lambda(s_2) = \lambda(s)$ et $\lambda(\Pi^k_{pt}) = \lambda(\Pi) + \lambda(s_1) = x^k_{pt}$. La séquence s_2 ainsi que les autres séquences restant dans $\Pi^k_{p,t-1}$ seront étendues avec un chemin différent de p durant les extensions non prioritaires. Si ce processus d'extension est répété pour chaque chemin p, alors une quantité maximale (au sens des coefficients) de séquences de chemins est étendue avec le chemin avec lequel elles se terminent déjà.

Autres extensions. Maintenant que toutes les extensions prioritaires ont été effectuées, considérons les séquences de chemins qui n'ont pas encore été étendues pour le pas de temps t. L'algorithme étend la première de ces séquences s avec n'importe quel chemin p tel que $\lambda(\Pi^k_{pt}) < x^k_{pt}$ avant de l'ajouter dans Π^k_{pt}. Comme pour les extensions prioritaires, si $\lambda(\Pi^k_{pt}) + \lambda(s) > x^k_{pt}$, pour obtenir exactement $\lambda(\Pi^k_{pt}) = x^k_{pt}$, la séquence s est divisée en deux parties s_1 et s_2 auxquelles seront affectés les coefficients suivant similaires à ceux pour les extensions prioritaires : $\lambda(s_1) = x^k_{pt} - \lambda(\Pi^k_{pt})$ et $\lambda(s_2) = \lambda(s) - \lambda(s_1)$. La première partie s_1 est étendue avec le chemin p avant d'être ajoutée à Π^k_{pt}. L'autre partie s_2 sera étendue plus tard par l'algorithme avec un autre chemin. L'algorithme étend ensuite une autre séquence. Une fois que toutes les séquences de chemins ont été étendues, les extensions peuvent commencer pour le pas de temps suivant.

Dans l'algorithme ci-dessus, les séquences de chemins sont étendues de sorte que la somme des coefficients des séquences de chemins utilisant le chemin p au pas de temps t soit x^k_{pt}. Ainsi, la combinaison convexe implique la même distribution de flot que les variables x^k_{pt}. Il reste à vérifier que les deux représentations de la distribution du flot impliquent les mêmes pénalités de changement de chemin. Dans l'algorithme, au pas de temps t, la somme des coefficients des séquences de chemins se terminant par le chemin p et prolongées par le chemin p est $\min(x^k_{pt}, x^k_{p,t-1})$. Toutes les autres séquences de chemins impliquent des pénalités de changement de chemin dont la somme est $x^k_{pt} - \min(x^k_{pt}, x^k_{p,t-1})$ ce qui est égal à $\max(0, x^k_{pt} - x^k_{p,t-1})$. C'est exactement la valeur prise par les variables n^k_{pt} qui modélisent les pénalités de changement de chemin dans la formulation par chemins étendue.

Avec l'algorithme ci-dessus, toute affectation des variables x^k_{pt} peut être traduite en une combinaison convexe de séquences de chemins induisant le même nombre de changements de chemin. Cela signifie que $Q_1 \subseteq Q_2$. Ainsi, la relaxation linéaire de la formulation par chemins étendue est aussi forte que la relaxation linéaire de la formulation par séquences de chemins. Ainsi, nous avons finalement montré que les deux relaxations linéaires sont équivalentes. □

La proposition ci-dessus montre que les deux formulations ont une relaxation linéaire de même qualité. Les deux formulations peuvent donc être utilisées indifféremment et le critère principal pour choisir l'une ou l'autre doit être le temps de résolution.

4.1.3 Formulation arc-nœud agrégée pour un pas de temps

La formulation arc-nœud agrégée s'inspire de la formulation arc-noeud présentée en Section 2.1.2 et modélise la relaxation linéaire d'un problème de flot insécable dynamique contenant un seul pas de temps. Cette formulation sera utilisée dans la suite de ce chapitre en conjonction avec l'heuristique SRR présentée au chapitre précédent pour créer des solutions du problème de flot insécable dynamique. Pour des raisons de lisibilité, nous supprimons l'index t indiquant le pas de temps considéré. Une pénalité est payée lorsqu'une commodité k n'utilise pas le chemin p^k utilisé au pas de temps précédent. Dans cette formulation, les commodités sont regroupées par origine pour créer des super-commodités. Une super-commodité k' contient plusieurs commodités k, et sa demande est la somme de leurs demandes entre leur origine commune et leurs destinations. Ce regroupement est fait pour réduire fortement le nombre de variables du modèle et donc le temps de résolution. Sans cette agrégation, il est difficile de calculer une relaxation linéaire à partir d'un modèle arc-nœud pour certaines grandes instances. Notons que cette formulation ne peut pas être convertie en un modèle de programmation linéaire en nombres entiers en raison du groupement des commodités. Les variables de cette formulation ont la signification suivante :

— $f_e^{k'}$ indique combien de flot la super-commodité k' envoie sur l'arc e ;

— $x^k_{p^k}$ décide de la proportion du flot de la commodité k qui est envoyé sur le chemin p^k utilisé au pas de temps précédent ;

— o_e représente le dépassement de capacité sur l'arc e ;

— o représente le dépassement de capacité qui dépasse le montant B du dépassement non pénalisé.

Puisqu'un seul pas de temps est considéré, si p^k est le chemin utilisé au le pas de temps précédent par la commodité k, la pénalité payée est égale à $1-x^k_{p^k}$. Ainsi, les pénalités peuvent être prises en compte dans l'objectif par le terme $\sum_{k\in K}(1-x^k_{p^k})$.

$$\min_{x^k_{p^k}, f_e^{k'}, o_e, o} \sum_{k\in K}(1-x^k_{p^k}) + o \tag{4.2a}$$

tel que

$$\sum_{e\in E^+(v)} f_e^{k'} - \sum_{e\in E^-(v)} f_e^{k'} = \sum_{k\in k'} d^k \delta_v^{O^k} - \sum_{k\in k'} d^k \delta_v^{D^k} \qquad \forall k'\in K',\ \forall v\in V \tag{4.2b}$$

$$\sum_{k\in k' | e\in p^k} x^k_{p^k} d^k \le f_e^{k'} \qquad \forall k'\in K',\ \forall e\in E \tag{4.2c}$$

$$\sum_{k'\in K'} f_e^{k'} \sum_{k\in k'} d^k \le c_e + o_e \qquad \forall e\in E \tag{4.2d}$$

$$\sum_{e\in E} o_e \le B + o \tag{4.2e}$$

$$f_e^{k'} \in \mathbb{R}^+, x^k_{p^k}\in[0,1],\ o_e\in\mathbb{R}^+,\ o\in\mathbb{R}^+ \qquad \forall k'\in K',\ \forall e\in E,\ \forall k\in K \tag{4.2f}$$

L'équation (4.2b) est la contrainte de conservation du flot pour la super-commodité k'. L'équation (4.2c) est une contrainte de liaison entre les variables de flot $f_e^{k'}$ et les variables de chemin $x_{p^k}^k$. Les équations (4.2d) et (4.2e) sont les contraintes de capacité : elles assurent que o_e représente le dépassement de capacité sur l'arc e et que o représente le dépassement de capacité au delà du seuil B.

Notons que cette formulation ne donne que le flot agrégé des commodités. Il est nécessaire de calculer exactement les chemins que chaque commodité utilise dans cette relaxation linéaire. Cela peut être effectué rapidement en $O(|V|(|E| + |K|))$ opérations à l'aide d'un algorithme de décomposition du flot (Ford Jr, 1956).

4.1.4 Restriction des chemins utilisables

Pour chaque commodité et à chaque pas de temps, un grand nombre de chemins sont utilisables. Si tous ces chemins sont pris en compte, le problème de flot insécable dynamique est difficile à résoudre en raison de son immense espace de solutions. Cependant, il n'est pas nécessaire de considérer tous les chemins pour obtenir des solutions de bonne qualité. Restreindre les chemins autorisés pour chaque commodité à un petit ensemble prédéfini simplifie le problème. Bien que la solution optimale globale puisse être perdue, il est plus facile de trouver des solutions optimales du problème restreint.

Les ensembles restreints de chemins utilisés dans ce travail sont calculés comme suit. Soit $P_{t\kappa}^k$ l'ensemble des κ-plus courts chemins pour la commodité k au pas de temps t lorsque qu'on considère une longueur unitaire pour chaque arc. Pour les méthodes considérant un seul pas de temps, l'ensemble restreint des chemins utilisables est défini égal à $P_{t\kappa}^k$. Pour les méthodes prenant en compte plusieurs pas de temps, l'ensemble restreint des chemins utilisables pour la commodité k au pas de temps t est égal à $\bigcup_{t' \in T} P_{t'\kappa}^k \cap P_t^k$ où P_t^k est l'ensemble de tous les chemins valides pour la commodité k au pas de temps t. De plus, les chemins initiaux sont également autorisés dans chaque pas de temps où ils sont valides. Cette restriction a déjà été utilisée dans des métaheuristiques conçues pour le problème de flot insécable par Laguna et Glover (1993) et Masri *et al.* (2015). De plus, elle nous permet d'utiliser des algorithmes *Branch and Bound* sans recourir à la méthode du *Branch and Price*. L'impact de cette restriction en termes de qualité de solution et de temps de calcul est étudié en Section 4.3.

4.2 Résolution de la formulation par séquences de chemins

Étant donné que la formulation par séquences de chemins présentée en Section 2.4 comporte un grand nombre de variables, un processus de génération de colonnes est requis pour résoudre sa relaxation linéaire. Nous commençons par présenter deux nouvelles approches pour résoudre le sous-problème de cette génération de colonnes. Puis, nous montrons comment utiliser l'heuristique SRR pour trouver des solutions binaires à partir de la relaxation

linéaire.

4.2.1 Génération de colonnes pour la formulation par séquences de chemins

Dans cette section, nous présentons la méthode de génération de colonnes proposée par Gamvros et Raghavan (2012) pour résoudre la formulation par séquence de chemins puis nous proposons deux méthodes alternatives pour trouver la variable de coût réduit minimum. Considérons la relaxation linéaire de la formulation par séquences de chemins du problème de flot insécable dynamique présentée en Section 2.4 :

$$\min_{x_s^k, o_{et}, o_t} \sum_{k \in K} \sum_{s \in S^k} n_s^k \; x_s^k + \sum_{t \in T} o_t \tag{4.3a}$$

tel que

$$\sum_{s \in S^k} x_s^k = 1 \qquad \forall k \in K \tag{4.3b}$$

$$\sum_{k \in K} \sum_{s \in S_{et}^k} x_s^k \; d^k \leq c_e(1 + o_{et}) \qquad \forall e \in E_t, \; \forall t \in T \tag{4.3c}$$

$$\sum_{e \in E_t} o_{et} \leq B + o_t \qquad \forall t \in T \tag{4.3d}$$

$$x_s^k \in \mathbb{R}^+, \; o_{et} \in \mathbb{R}^+, \; o_t \in \mathbb{R}^+ \qquad \forall s \in \bigcup_k S^k, \; \forall k \in K, \; \forall e \in E_t, \; \forall t \in T \tag{4.3e}$$

Cette formulation possède une variable x_s^k pour chaque séquence de chemins valide et pour chaque commodité. Cela représente un nombre exponentiel de variables en le nombre d'arcs, de nœuds et de pas de temps. Ce très grand nombre de variables explique que cette formulation soit exclusivement résolue à l'aide d'une méthode de génération de colonnes. Trouver la variable de coût réduit minimum est une partie de essentielle de la génération de colonnes. Pour plus de détails sur la méthode de la génération de colonnes, se référer aux annexes. Afin de pourvoir calculer le coût réduit d'une variable x_s^k, détaillons les coefficients lui étant associés dans la fonction objectif et dans les différentes contraintes.

Premièrement, le coût associé à une variable x_s^k dans la fonction objectif est le nombre n_s^k de changements de chemin induit par la séquence de chemins s. Cette variable possède aussi un coefficient unitaire dans la contrainte (4.3b) associée à la commodité k. Dans les contraintes (4.3d), le coefficient des variables x_s^k est nul. Enfin, cette variable apparaît avec un coefficient d^k dans toutes les contraintes de type (4.3c) associées à l'un des arcs présents dans les chemins de la séquence s. Ainsi, une variable x_s^k où $s = (p_1, ..., p_{|T|})$ a un coût réduit égal à :

$$cr_s^k = n_s^k - u^k - d^k \sum_{t \in T} \sum_{e \in p_t} u_{et}$$

où u^k est la variable duale de la contrainte (4.3b) associée à la commodité k et u_{et} est la

variable duale de la contrainte (4.3c) associée à l'arc e au pas de temps t.

Le problème de minimisation du coût réduit sur l'ensemble des séquences de chemins d'une commodité k est donc le suivant : choisir la séquence de chemins avec le plus petit coût tel que choisir un chemin p_t au pas de temps t induit un coût $-d^k \sum_{e \in p_t} u_{et}$ et ne pas garder le même chemin de t à $t+1$ induit un coût unitaire. Notons que pour une commodité fixée, le terme $-u^k$ est constant et a été retiré du problème. Nous présentons maintenant la méthode proposée par Gamvros et Raghavan (2012) afin de résoudre ce problème.

Gamvros et Raghavan (2012) proposent de décomposer le problème en deux parties. Tout d'abord, restreindre l'ensemble des chemins à considérer à un sous-ensemble $\hat{P}_t^k$ pour chaque pas de temps. Deuxièmement, en utilisant les chemins présents dans $\hat{P}_t^k$, construire une séquence de chemins de moindre coût réduit. Cette deuxième partie peut être résolue grâce à la programmation dynamique. En effet, le choix du meilleur chemin pour un pas de temps t dépend du chemin choisi au pas de temps $t-1$ mais pas des chemins choisis avant $t-1$. Pour plus de détails sur l'algorithme de programmation dynamique, nous renvoyons le lecteur aux travaux de Gamvros et Raghavan (2012).

L'ensemble $\hat{P}_t^k$ de chemins considérés ne peut pas être l'ensemble P_t^k des chemins valides pour la commodité k au temps t car la cardinalité de P_t^k est exponentielle en la taille du graphe. Gamvros et Raghavan (2012) définissent $\hat{P}_t^k$ comme le résultat d'un algorithme de k-plus-courts chemins sur le graphe $G_t = (V, E_t)$, où le coût de chaque arc $e \in E_t$ est égal à $-d^k u_{et}$. Notons les chemins renvoyés $p_1, ..., p_\kappa$ où p_1 est le plus court chemin et p_κ le plus long chemin renvoyé par l'algorithme. Le nombre de chemins κ calculé par l'algorithme de k-plus-courts chemins est fixé tel que $c(p_\kappa) \geq c(p_1) + 2$ où $c(p)$ désigne le coût d'un chemin p et la valeur 2 est le coût de deux pénalités de changement de chemin. Autrement dit, il est garanti que tous les chemins ayant une différence de coût inférieur à 2 ont été calculés. Or, une séquence de chemins de coût réduit minimum peut être calculée en utilisant uniquement les chemins vérifiant cette condition. Ainsi, supposons qu'une séquence de chemins de coût réduit minimum utilise au pas de temps t un chemin p ne vérifiant pas $c(p) \leq c(p_1) + 2$. On peut alors remplacer p par p_1 pour créer une séquence de chemins de coût réduit inférieur. En effet, l'échange ne peut augmenter que de deux le nombre de pénalités induit par la séquence de chemins et cette augmentation est compensée par une diminution du coût du chemin d'un montant d'au moins deux. La nouvelle séquence de chemins est donc bien de coût réduit minimum et utilise l'un des chemins renvoyé par l'algorithme de k-plus-court chemins.

Une fois la meilleure séquence de chemins calculée pour chaque commodité, toutes les variables de coût réduit négatif calculées sont ajoutées au problème maître qui calcule alors une nouvelle solution primale-duale.

La méthode proposée par Gamvros et Raghavan (2012) est basée sur des calculs de k-plus-courts chemins avec une condition d'arrêt basée sur la différence de coût entre le chemin calculé le plus court et le plus long. Cette méthode présente l'inconvénient suivant : la présence d'arcs de coût nul lors du calcul des k-plus-courts chemins peut induire une infinité de plus courts chemins équivalents. Plus précisément, il existe deux versions du problème des k-plus-courts chemins. Lorsque les chemins renvoyés sont autorisés à avoir des cycles, un cycle de coût nul

dans le graphe peut générer un nombre infini de chemins de même coût. En revanche, lorsque le chemin renvoyé ne doit pas contenir de cycles, il existe un nombre fini de chemins de même longueur mais ce nombre peut être exponentiel en la taille du graphe. Par exemple, cela peut se produire si un très grande nombre d'arcs ont un coût nul. Pour éviter cet inconvénient, nous présentons de nouvelles méthodes pour résoudre le problème de *pricing* n'étant pas basées sur des calculs de k-plus-courts chemins.

***Pricing* sans k-plus-courts chemins.** Dans une séquence de chemins, notons $t_1...t_2$ toute séquence de pas de temps consécutifs telle que la séquence de chemins change de chemin avant t_1 et après t_2, mais pas entre t_1 et t_2. Soit $p_{t_1...t_2}$ le chemin donnant le coût total le plus bas sur cette séquence s'il existe. La méthode est basée sur l'énoncé suivant. Le chemin $p_{t_1...t_2}$ est indépendant des chemins choisis en dehors de $t_1...t_2$ et peut être calculé avec un seul appel à un algorithme de plus court chemin sur un graphe $G_{t_1...t_2}$. Les nœuds de $G_{t_1...t_2}$ sont $V_{t_1...t_2} = V$, ses arcs sont $E_{t_1...t_2} = \bigcap_{t=t_1}^{t_2} E_t$ et la longueur d'un arc $e \in E_{t_1...t_2}$ est $l_e = d^k \sum_{t=t_1}^{t_2} u_{et}$. Notons qu'il peut n'y avoir aucun chemin valide pour une commodité sur $G_{t_1...t_2}$. Dans ce cas, la commodité doit changer de chemin entre t_1 et t_2. Soit $\hat{P}_t^k$ l'ensemble des chemins considéré par l'algorithme de programmation dynamique de Gamvros et Raghavan (2012). L'algorithme de programmation dynamique calcule la séquence de chemins de coût réduit minimum par l'ensemble des séquences de chemins n'utilisant que les chemins de $\hat{P}_t^k$. Si l'on pose $\hat{P}_t^k = \{p_{t_1...t_2},\ \forall t_1...t_2 \text{ tel que } t \in t_1...t_2\}$ alors une séquence de chemins de coût réduit minimum peut être calculée en utilisant uniquement les chemins de $\hat{P}_t^k$. En effet, supposons qu'une séquence de chemins de coût réduit minimum utilise au pas de temps t un chemin p qui n'est pas dans $\hat{P}_t^k$. Le chemin p est conservé sur une séquence $t_1...t_2$ et peut être remplacé par $p_{t_1...t_2}$ sur cette séquence pour créer une nouvelle séquence de chemins de coût réduit minimum.

Cette nouvelle méthode pour calculer les ensembles $\hat{P}_t^k$ utilise $\frac{|T|(|T|-1)}{2}$ appels un algorithme du plus court chemin où $|T|$ est le nombre de pas de temps considéré. Par rapport à la méthode de Gamvros et Raghavan (2012), le nombre de chemins calculés est quadratique en le nombre de pas de temps au lieu de linéaire. Cependant, dans cette méthode, le nombre de chemins calculés est constant en la taille du graphe, alors que la méthode de Gamvros et Raghavan (2012) peut calculer un nombre exponentiel de chemins en la taille du graphe. La nouvelle méthode utilise toujours l'algorithme de programmation dynamique de Gamvros et Raghavan (2012) pour calculer la meilleure séquence de chemins. Nous décrivons maintenant une méthode de résolution du problème de *pricing* sans programmation dynamique.

***Pricing* sans programmation dynamique.** Dans cette méthode, toutes les sous-séquences de pas de temps consécutifs sont considérées par ordre de longueur croissante. La séquence de chemins optimale est d'abord calculée pour les sous-séquences de longueur un : pour une séquence $t_1...t_1$ c'est le plus court chemin du graphe $G_{t_1} = (V, E_{t_1})$ ayant des coûts $-d^k u_{et_1}$ sur ses arcs. Ensuite, supposons que la séquence de chemins optimale a été calculée pour toutes les sous-séquences de longueur inférieure à n. La séquence de chemins optimale pour une sous-séquence $t_1...t_2$ de longueur $n+1$ est calculée comme suit. Une disjonction de cas est faite : soit la séquence de chemins ne fait aucun changement de chemin, soit elle fait un changement de chemin après un certain pas de temps t^*. Dans le premier

cas, la séquence de chemins optimale est d'utiliser le chemin $p_{t_1...t_2}$ sur toute la séquence. Voir la définition et la construction de $p_{t_1...t_2}$ dans le paragraphe "*Pricing* sans k-plus-courts chemins". Dans tous les autres cas, puisqu'un changement de chemin est effectué après le pas de temps t^*, la séquence de chemins optimale pour la séquence $t_1...t_2$ est la juxtaposition des séquences de chemins optimales pour les séquences $t_1...t^*$ et $t^* + 1...t_2$ qui ont déjà été calculées. L'algorithme s'arrête lorsque la séquence de chemins optimale a été calculée pour l'unique séquence de taille $|T|$ qui est celle qui contient tous les pas de temps.

Nous étudions maintenant la complexité de cette dernière méthode. Notons $|E| = \max_t |E_t|$. Pour une sous-séquence $t_1...t_2$, dans le cas où aucun changement de chemin n'est effectué, la méthode commence par calculer $G_{t_1...t_2}$. Puisque $G_{t_1...t_2}$ a les mêmes nœuds que $G_{t_1...t_2-1}$ et ses arcs sont $E_{t_1...t_2} = E_{t_1...t_2-1} \cap E_{t_2}$, alors $G_{t_1...t_2}$ peut être calculé à partir de $G_{t_1...t_2-1}$ et G_{t_2} en $O(|E_{t_1...t_2-1}|)$ opérations ce qui est inférieur à $O(|E|)$ opérations. En utilisant un algorithme de Dijkstra, le calcul de $p_{t_1...t_2}$ à partir de $G_{t_1...t_2}$ prend $O(|E_{t_1...t_2}| \ \ln |E_{t_1...t_2}|)$ opérations ce qui est aussi $O(|E| \ \ln |E|)$ opérations. Les autres $t_2 - t_1$ cas où un changement de chemin est effectué sur la sous-séquence $t_1...t_2$ peuvent être traités en temps constant. Puisqu'il y a $\frac{|T|(|T|-1)}{2} = O(|T|^2)$ différentes sous-séquences, la complexité totale de la méthode est $O(|T|^3 + |T|^2|E| \ln |E|)$. La méthode utilise le même nombre d'appels à un algorithme de plus court chemin que la méthode précédente : $\frac{|T|(|T|-1)}{2}$.

Pour la clarté de la présentation, nous avons décrit les méthodes de *pricing* comme si aucun chemin initial n'était donné pour chaque commodité. Pour tenir compte des chemins initiaux, on ajoute simplement les chemins initiaux dans l'ensemble des chemins considérés $\hat{P}_t^k$ dans les méthodes utilisant l'algorithme de programmation dynamique de Gamvros et Raghavan (2012). Dans la dernière méthode, une disjonction de cas supplémentaire doit être faite sur la durée durant laquelle le chemin initial est conservé.

4.2.2 L'heuristique SRR et la génération de colonnes

Maintenant que la relaxation linéaire de la formulation par séquences de chemins peut être résolue, l'heuristique d'arrondi aléatoire séquentiel (SRR), présentée en Section 3.1, peut être utilisée pour trouver une solution entière. Une application directe de l'heuristique recommanderait d'effectuer toutes les itérations de génération de colonnes nécessaires pour trouver la solution linéaire optimale avant d'appliquer les étapes d'arrondi aléatoire. Cependant, comme l'algorithme SRR est une heuristique, il est raisonnable de travailler avec une approximation de la solution linéaire. Cela signifie que nous pouvons n'effectuer que quelques itérations de génération de colonnes avant d'appliquer les étapes d'arrondi aléatoire. En retour, nous pouvons actualiser la relaxation linéaire plus souvent.

Dans un premier temps, un grand nombre d'itérations de génération de colonnes sont effectuées pour obtenir une bonne approximation de la relaxation linéaire. Ensuite, l'heuristique SRR est appliquée en alternant deux types d'actions :

— actualiser la relaxation linéaire via des itérations de génération de colonnes ;

— choisir un chemin pour une commodité et un pas de temps par arrondi aléatoire.

4.3 Étude expérimentale

Dans cette section, nous rapportons une comparaison expérimentale de différents solveurs basés sur les concepts présentés dans les sections précédentes. Plusieurs aspects clés des solveurs expliquant leurs performances sont mis en évidence à travers une étude par ablation. Nous présentons d'abord comment les instances sont générées. Ensuite, les différents solveurs et leurs paramètres sont détaillés. Enfin, les résultats expérimentaux sont présentés et discutés. Les jeux de données et le code utilisés dans cette partie expérimentale sont accessibles à l'adresse `https://github.com/SuReLI/Dynamic_mcnf_paper_code.git`. Le code de ce travail a été écrit en Python 3 et utilise le solveur commercial Gurobi Optimization (2020) dans sa version 8.11. Les expériences ont été faites sur un serveur ayant 48 CPU Intel Xeon E5-2670 2.30 GHz, 60 Gbit de RAM et CentOS Linux 7.

4.3.1 Génération des instances

La création d'une instance du problème de flot insécable dynamique comprend quatre étapes :

1. Création du graphe initial ;
2. Création de la liste initiale des commodités ;
3. Création du chemin initial de chaque commodité ;
4. Pour chaque pas de temps, le graphe et la liste des commodités du pas de temps précédent sont modifiés pour en créer de nouveaux pour ce pas de temps.

Graphe et liste de commodités initiaux. Le graphe initial et la liste des commodités des instances sont créés avec la méthode présentée en Section 3.4 pour les instances du problème de flot insécable statique.

Notons qu'une liste de commodités créée de cette manière peut toujours être acheminée dans les capacités des arcs en utilisant, pour chaque commodité, le chemin utilisé pour créer la commodité. Cependant, ce n'est vrai que pour le premier pas de temps. Lors des pas de temps suivants, à cause des modifications apportées aux commodités et au graphe, il n'y a aucune garantie que toutes les commodités puissent être acheminées en respectant les capacités. Dans ce chapitre, nous utiliserons les deux méthodes décrites en Section 3.4 pour choisir la demande des commodités : $d = U(\min(c_p, \hat{d}_{max}))$ et $d = \min(c_p, U(\hat{d}_{max}))$. Ceci nous permet de considérer des instances plus ou moins dures pour une même taille de graphe.

Liste des chemins initiaux. La liste des chemins initiaux est choisie pour être la liste des chemins utilisés pour créer les commodités.

Pas de temps suivants. Une instance du problème de flot insécable dynamique est une séquence de problèmes de flot insécable représentant différents pas de temps. Chaque pas de temps introduit quelques changements au problème : certaines commodités changent d'origine ou de destination, certains arcs sont ajoutés ou supprimés. Les modifications suivantes sont apportées entre chaque pas de temps dans nos tests :

— Pour chaque commodité (O, D, d), avec une probabilité μ, la destination D de la commodité est remplacée par un nœud u qui n'est pas l'origine d'une commodité et tel que $(D, u) \in E_t$. Si aucun nœud de ce type n'existe, la destination reste inchangée.

— Pour chaque arc $e = (O, u)$ partant d'une origine O, avec probabilité μ, l'arc e est remplacé par un arc $e' = (O, v)$ où v est un nœud qui n'est pas à l'origine d'une commodité et tel que $(v, u) \in E_t$. Si aucun nœud de ce type n'existe, aucun changement ne se produit.

La probabilité μ d'effectuer un changement est fixée à $\mu = 0.03$ afin de ressembler aux instances issues de la constellation présentée au Chapitre 1. Notons qu'en ne changeant que les destinations des commodités, le nombre d'origines différentes reste faible. Notons également qu'avec ces changements d'arcs, un graphe fortement connexe reste fortement connexe.

Les jeux de données. Quatre jeux de données différents sont utilisés au cours des expériences. Trois d'entre eux considèrent des graphes de tailles différentes tandis que le dernier considère des graphes de taille fixe mais un nombre variable de commodités. Dans chaque jeu de données, le montant de dépassement de capacité autorisé B à chaque pas de temps est égal à 1% de la demande totale des commodités. Dans chaque jeu de données, un paramètre varie et dix instances sont générées pour chaque valeur de ce paramètre.

— Jeu de données *grille facile* : ce jeu de données considère les graphes grille de 12 nœuds à 156 nœuds. Lors du choix de la demande des commodités, la formule $d = U(\min(c_p, \hat{d}_{max}))$ est utilisée, créant ainsi de nombreuses petites commodités, ce qui rend les instances plus faciles à résoudre. Les capacités des arcs de la grille sont fixées à 15 000 tandis que les capacités des arcs supplémentaires sont fixées à 10 000. Ceci facilite également la résolution des instances.

— Jeu de données *grille difficile* : ce jeu de données considère les graphes grille de 6 nœuds à 90 nœuds. Lors du choix de la demande des commodités, la formule $d = \min(c_p, U(\hat{d}_{max}))$ est utilisée.

— Jeu de données *aléatoire connexe* : ce jeu de données considère des graphes aléatoires fortement connexes de 12 nœuds à 182 nœuds. Lors du choix de la demande des commodités, la formule $d = \min(c_p, U(\hat{d}_{max}))$ est utilisée.

— Jeu de données *taille des commodités* : ce jeu de données considère des capacités des arcs uniformes c_{et} allant de 1 à 1000 selon l'instance, tandis que le paramètre $\hat{d}_{max}$ est choisi égal à $\sqrt{c_{et}}$. Ceci induit un nombre variable de commodités ainsi que des commodités de tailles différentes par rapport aux capacités des arcs. Tous les graphes considérés sont des graphes grille de 42 nœuds. Lors du choix de la demande des commodités, la formule $d = \min(c_p, U(\hat{d}_{max}))$ est utilisée.

Notons que les instances des jeux de données ci-dessus sont plus petites (maximum 182 noeuds) que les instances induites par notre application industrielle : la constellation Telesat contient 300 satellites et 100 stations-sol ce qui induit un total de 400 noeuds. La raison de ce choix est que nous voulons pouvoir lancer les solveurs les plus lourds sur un grand nombre d'instances. En revanche, les solveurs les plus rapides que nous présentons sont capables de résoudre l'instance de la constellation Telesat en un temps raisonnable.

4.3.2 Les solveurs

Dans cette section, nous présentons les solveurs utilisés dans les expériences. Nous commençons par les solveurs basés sur des formulations considérant un seul pas de temps.

SRR-arc-nœud. Dans cette méthode, l'heuristique SRR est appliquée à la formulation arc-nœud agrégée de la Section 4.1.3 avec la pénalisation du flot présentée en Section 4.3.3. Les commodités ayant la plus grande demande sont arrondies en premier.

SRR-chemin. Cette méthode consiste à appliquer l'heuristique SRR à la formulation par chemins étendue de la Section 4.1.1 où un seul pas de temps est considéré. La pénalisation du flot présentée en Section 4.3.3 est utilisée et les commodités ayant la plus grande demande sont arrondies en premier.

SRR-arc-nœud-sans-pénalisation. Similaire à "SRR-arc-nœud" ; cependant, la pénalisation du flot n'est pas utilisée.

SRR-chemin-sans-pénalisation. Similaire à "SRR-chemin" ; cependant, la pénalisation du flot n'est pas utilisée.

Nous décrivons maintenant les solveurs obtenus lorsque l'ensemble des chemins autorisés pour chaque commodité est restreint tel que présenté en Section 4.1.4. Ces solveurs n'utilisent que la formulation par chemins étendue où un seul pas de temps est considéré.

B&B-restreint-court/moyen/long. Dans cette méthode, une procédure standard *Branch and Bound* est appliquée à la formulation par chemins étendue où un seul pas de temps est considéré et où l'ensemble des chemins autorisés pour chaque commodité est restreint. La procédure *Branch and Bound* se poursuit jusqu'à ce qu'une limite de temps soit atteinte. En pratique, la limite de temps est fixée à $\tau = \alpha 1.7^{\sqrt{|V|}}$ pour obtenir une croissance effective du temps de calcul similaire aux autres méthodes. Le coefficient α est fixé à : $1/50$ pour la version courte, $1/10$ pour la version moyenne, $1/2$ pour la version longue.

SRR-restreint. Similaire à "SRR-chemin" ; cependant, l'ensemble des chemins autorisés pour chaque commodité est restreint tel que présenté en Section 4.1.4.

Enfin, nous présentons les solveurs basés sur la formulation par séquences de chemins présentée en Section 2.4. Ces solveurs diffèrent par l'utilisation de la pénalisation du flot, l'ensemble de chemins utilisables pour chaque commodité et leur ordre d'arrondi.

SRR-séquence. SRR est appliqué à la formulation par séquences de chemins avec la pénalisation du flot présentée en Section 4.3.3. Lors d'une étape d'arrondi aléatoire, pour une commodité, une séquence de chemins est choisie parmi celles présentes dans la relaxation linéaire par arrondi aléatoire.

SRR-séquence-sans-pénalisation. Similaire à "SRR-séquence" ; cependant, la pénalisation du flot n'est pas utilisée.

SRR-séquence-restreint. Similaire à "SRR-séquence" ; cependant, l'ensemble des chemins utilisables pour chaque commodité est restreint comme en Section 4.1.4.

SRR-séquence-pas-de-temps. Similaire à "SRR-séquence" ; cependant, au lieu d'arrondir une séquence de chemins à chaque itération, les chemins du premier pas de temps sont arrondis consécutivement, puis ceux du deuxième pas de temps, etc. À l'intérieur d'un pas de temps, l'algorithme commence par les commodités ayant la plus grande demande.

4.3.3 Réglages des paramètres

Dans cette section, nous décrivons quelques hyper-paramètres supplémentaires de nos algorithmes ainsi que leur valeur dans les expériences.

Seuil θ de SRR. Le seuil déterminant la fréquence d'actualisation de la relaxation linéaire est fixé à $\theta = |V|$ pour les méthodes basées sur la formulation par séquences de chemins et $|V|/10$ sinon.

Pénalisation du flot. Un coût supplémentaire est ajouté aux variables représentant le flot dans les différents modèles. Pour les variables représentant un chemin p, ce coût supplémentaire est $|E(p)|\epsilon$ où $|E(p)|$ est le nombre d'arcs dans le chemin p et ϵ est une petite constante fixée à 10^{-4} dans nos expériences. Pour les variables de séquences de chemins représentant une séquence de chemins $p_1, ..., p_T$, le coût supplémentaire est $\sum_{t \leq T} |E(p_t)|\epsilon$. Pour la formulation arc-nœud agrégée, les variables de flot associées à chaque arc se voient attribuer un coût ϵ. Dans nos tests préliminaires, l'ajout ou non de la pénalisation du flot n'est apparue importante que lorsque l'ensemble des chemins autorisés pour chaque commodité n'est pas restreint, la distinction n'est donc faite que dans ce cas.

Suppression de variables. Lors de la génération de colonnes dans les formulations par chemins ou par séquences de chemins, les modèles ont tendance à accumuler un grand nombre de variables inutiles (variables non utilisées dans la solution optimale actuelle) ce qui augmente le temps de résolution du modèle. Pour éviter cela, chaque fois que le modèle est résolu, chaque variable hors-base de la solution optimale est supprimée avec une probabilité de 0.3.

Taille des ensembles de chemins restreints. Certains des solveurs ne considèrent qu'un nombre restreint de chemins pour chaque commodité, comme expliqué en Section 4.1.4. Le nombre de k-plus-courts chemins calculés est de 4 dans nos tests.

Nombre d'itérations de génération de colonnes. Les solveurs basés sur la formulation par séquences de chemins n'effectuent pas toutes les itérations de génération de colonnes nécessaires pour obtenir la solution optimale de la relaxation linéaire. Cela diminue le temps de calcul global des algorithmes. Avant la première étape d'arrondi aléatoire, 20 itérations de génération de colonnes sont effectuées pour obtenir une bonne approximation de la relaxation linéaire. Cependant, entre les étapes d'arrondi aléatoire, seules 3 itérations de génération de colonne sont effectuées. Faire plus d'itérations est apparu inutile dans nos tests préliminaires.

Méthode de *pricing* pour la formulation par séquences de chemins. Pour résoudre la formulation par séquences de chemins, un processus de génération de colonnes doit être utilisé. Celui ci repose sur une méthode pour calculer les variables ayant un coût réduit minimum. Trois méthodes de *pricing* ont été présentées en Section 4.2.1. La première méthode, introduite par Gamvros et Raghavan (2012) n'est pas applicable dans notre contexte en raison des limitations présentées dans la même section. La méthode de *pricing* utilisée est celle présentée dans le paragraphe "*Pricing* sans k-plus-courts chemins" utilisant l'algorithme de programmation dynamique introduit par Gamvros et Raghavan (2012).

Paramètres des *Branch and Bound*. Le paramètre `MIPFocus` de Gurobi Optimization (2020) a été choisi égal à 1 pour encourager le solveur à trouver rapidement des solutions de bonne qualité. De plus, pour effectuer une comparaison équitable avec les autres solveurs, ces solveurs ont été limités à l'utilisation d'un seul CPU. Cela empêche les solveurs utilisant le *Branch and Bound* d'utiliser les dizaines de CPU disponibles sur le serveur.

4.3.4 Calcul du nombre minimum de changements de chemin

En raison des modifications apportées aux instances entre chaque pas de temps, un chemin valide pour une commodité au pas de temps t peut ne pas être valide pour la même commodité au pas de temps $t+1$. Ainsi, même si les capacités du graphe étaient infinies, le nombre minimum de changements de chemin dans une solution valide serait presque toujours supérieur à zéro. Pour interpréter correctement les solutions fournies par les algorithmes testés, nous devons connaître le nombre minimum de changements de chemin réalisables lorsque le dépassement de capacité n'est pas pénalisé. Lorsque qu'il n'est pas pénalisé, la solution peut être calculée indépendamment pour chaque commodité. L'Algorithme 2 décrit comment construire une solution optimale pour une commodité. Cet algorithme commence avec le chemin initial donné et le conserve jusqu'à ce qu'il devienne invalide. Ensuite, il trouve un chemin qui peut être conservé le plus longtemps possible et le conserve jusqu'à ce qu'il devienne invalide. Ce processus continue jusqu'à ce qu'un chemin pour tous les pas de temps ait été choisi.

Algorithme 2 Calcul d'une séquence de chemins ayant nombre minimum de changements de chemin

Entrées : $G = (V, E, (E_t)_{t\in T})$ un graphe dynamique, $(O_t, D_t, d_t)_{t\in T}$ une commodité, p_{init} un chemin initial de la commodité

1: Affecter $p^* = p_{init}$ et $t_{min} = 1$
2: Calculer $t_{max}(p^*, t_{min}) = \max\{t | \forall t' \in \{t_{min}, ..., t\}, p^*$ est valide pour $G_{t'}$ et $(O_{t'}, D_{t'}, d_{t'})\}$
3: Pour tout $t \leq t_{max}(p^*, t_{min})$, affecter $p_t = p^*$
4: Affecter $t_{min} = t_{max}(p^*, t_{min}) + 1$
5: **Tant que** $t_{min} \leq |T|$ **faire**
6: Affecter $p^* = \text{argmax}_p\ t_{max}(p, t_{min})$
7: Pour tout $t \in \{t_{min}, ..., t_{max}(p^*, t_{min})\}$, affecter $p_t = p^*$
8: Affecter $t_{min} = t_{max}(p^*, t_{min}) + 1$
9: **Renvoyer** $(p_t)_{t\in T}$

Théorème 4.1
La séquence de chemins renvoyée par l'Algorithme 2 ne contient pas plus de changements de chemin que toute autre séquence de chemins utilisant uniquement des chemins valides à chaque pas de temps.

Démonstration. Pour prouver le théorème, nous partirons de n'importe quelle solution utilisant un nombre minimum de changements de chemin et montrerons que nous pouvons répliquer la solution renvoyée par l'algorithme 2 en appliquant des remplacements sur cette solution. La preuve repose sur le fait que ces remplacements n'augmentent pas le nombre de changements de chemin, ce qui implique que la solution renvoyée par l'algorithme utilise également un nombre minimum de changements de chemin.

Nous utilisons les deux remplacements suivants :

1. Si une solution utilise un chemin p au pas de temps t qui est valide pour le pas de temps $t+1$, mais utilise un autre chemin p' au pas de temps $t+1$, alors on remplace le chemin p' par le chemin p au pas de temps $t+1$.
2. Si une solution utilise un chemin p sur une sous-séquence $t_1...t_2$ alors on remplace p sur $t_1...t_2$ par un chemin valide pour un nombre maximum de pas de temps contigus après t_1.

Les remplacements sont utilisés comme suit. Les chemins sont remplacés du premier au dernier. Lorsque l'on considère le remplacement pour un pas de temps t, si le chemin utilisé dans le pas de temps $t-1$ est valide au pas de temps t alors on utilise le remplacement 1, sinon, on utilise le remplacement 2. Dans les deux cas, le nouveau chemin est le même que celui utilisé par l'algorithme puisque ces remplacements imitent la façon dont l'algorithme crée sa solution. □

Pour décrire les résultats de nos expériences, la qualité de la solution en termes de changements de chemin sera donnée comme le rapport du nombre de changements de chemin dans la solution sur le nombre minimum de changements de chemin de toute solution valide. Ce rapport est appelé ratio de changements de chemin.

4.3.5 Résultats expérimentaux

Dans cette section, chaque figure présente un sous-ensemble d'algorithmes sur l'un des quatre jeux de données pour l'une des trois métriques suivantes : temps de calcul, ratio de dépassement, ratio de changements de chemin. Notons que bien que tous les solveurs aient été évalués sur tous les jeux de données, par souci de lisibilité, certaines figures ne sont présentées qu'en annexe. Le ratio de changements de chemin est présenté en Section 4.3.4 et la valeur utilisée dans les figures est le ratio de changements de chemin moins un. En effet, ceci permet de mieux mettre en évidence (à l'aide d'une échelle logarithmique) la qualité des solveurs qui donnent un ratio de changements de chemin proche de un. Le ratio de dépassement est calculé comme suit. A chaque pas de temps, un montant B de dépassement

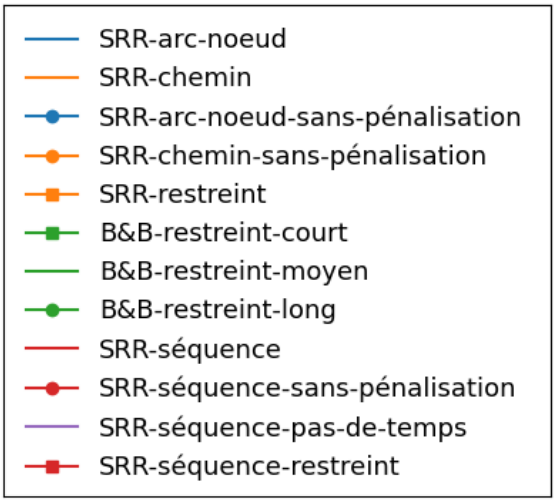
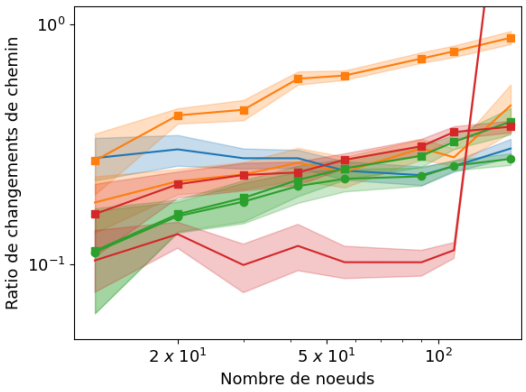


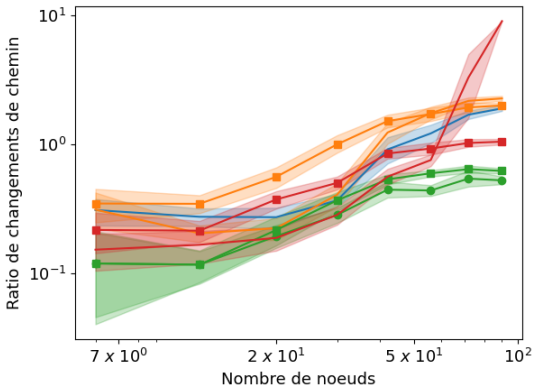
FIGURE 4.1 – Légende des figures 4.2 à 4.5

de capacité est autorisé sans pénalisation. Une solution peut dépasser ce montant autorisé et avoir des dépassements pénalisés o_t à certains pas de temps. Le ratio de dépassement est le dépassement total pénalisé d'une solution sur le dépassement total autorisé d'une instance : $\frac{\sum_{t \in T} o_t}{|T|B}$. Le ratio de dépassement s'étend sur plusieurs ordres de grandeur tout en prenant également des valeurs nulles. Pour afficher correctement cette métrique, nous utilisons une échelle logarithmique symétrique. Cette échelle est logarithmique sauf autour de zéro où elle est linéaire. Le temps de calcul est donné en secondes. Lorsqu'un algorithme dépasse une limite de temps de 3 heures, il est arrêté et se voit affecter des résultats ad hoc mauvais : temps de calcul 3 heures, ratio de changements de chemin 10 et ratio de dépassement 10. Cela arrive à quelques algorithmes basés sur la formulation par séquences de chemins sur les plus grandes instances. Dans chaque figure, les résultats des instances créées à l'aide des mêmes paramètres sont agrégés. Les courbes tracées représentent les résultats moyens sur les instances agrégées, tandis que les intervalles de confiance à 95% pour la moyenne sont représentés en semi-transparence autour de la courbe principale. Ces intervalles de confiance sont créés à l'aide de la méthode statistique du Bootstrap (Efron, 1992) avec un nombre de rééchantillonnages égal à 1000. Une légende commune pour les figures de cette section est donnée en Figure 4.1.

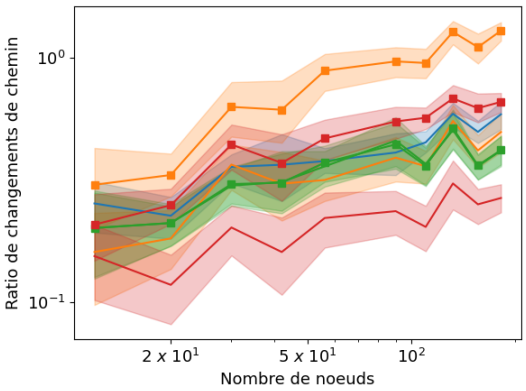
Restriction des chemins autorisés. Afin d'analyser l'influence de la restriction de l'ensemble des chemins autorisés présentée en Section 4.1.4, nous comparons SRR-chemin à SRR-restreint et SRR-séquence à SRR-séquence-restreint dans les figures 4.2, 4.3 et 4.4. Restreindre les chemins autorisés à un sous-ensemble de k-plus-courts chemins a les effets suivants. Dans la plupart des cas, le temps de calcul est réduit, notamment pour les grandes instances (typiquement de 2 à 5 fois mais parfois de plusieurs ordres de grandeur sur les instances difficiles). Dans de nombreux cas, la restriction des chemins autorisés augmente le



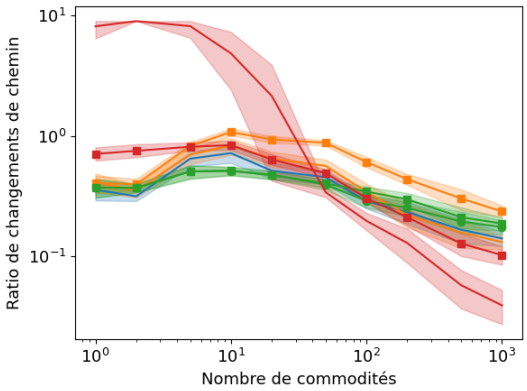
(a) Jeu de données *Grille facile*



(b) Jeu de données *Grille difficile*

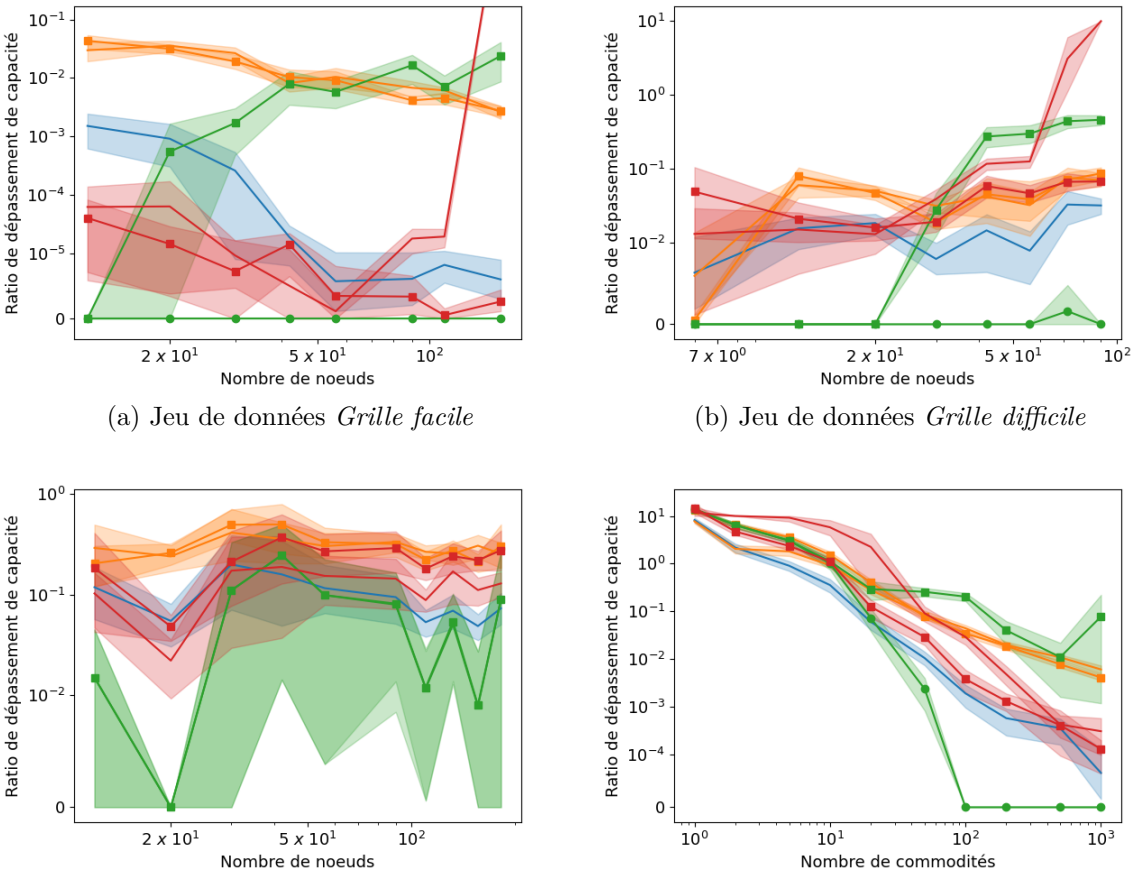


(c) Jeu de données *Aléatoire connexe*



(d) Jeu de données *Taille de commodités*

FIGURE 4.2 – Ratio de changement de chemin pour les algorithmes principaux sur tous les jeux de données



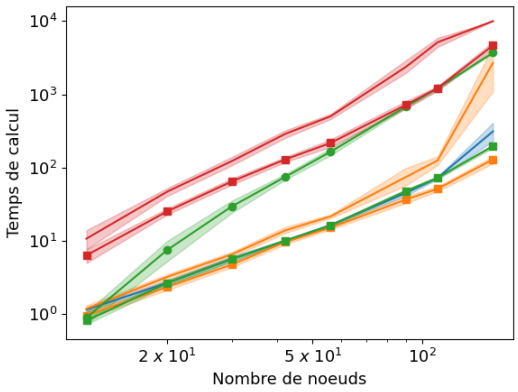
(a) Jeu de données *Grille facile*

(b) Jeu de données *Grille difficile*

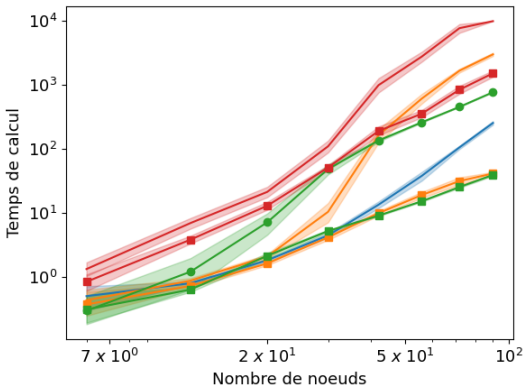
(c) Jeu de données *Aléatoire connexe*

(d) Jeu de données *Taille de commodités*

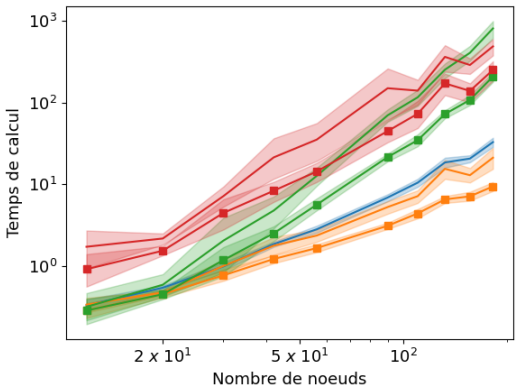
FIGURE 4.3 – Ratio de dépassement pour les algorithmes principaux sur tous les jeux de données



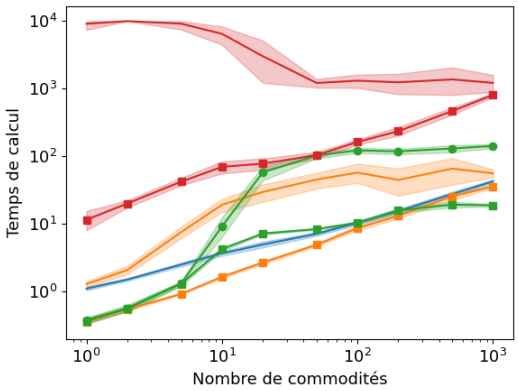
(a) Jeu de données *Grille facile*



(b) Jeu de données *Grille difficile*

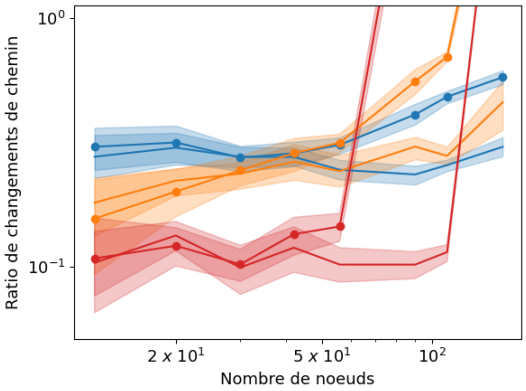


(c) Jeu de données *Aléatoire connexe*

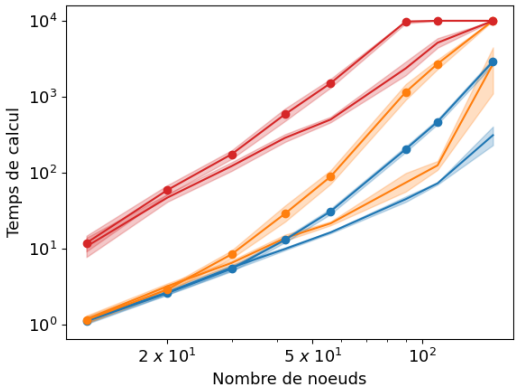


(d) Jeu de données *Taille de commodités*

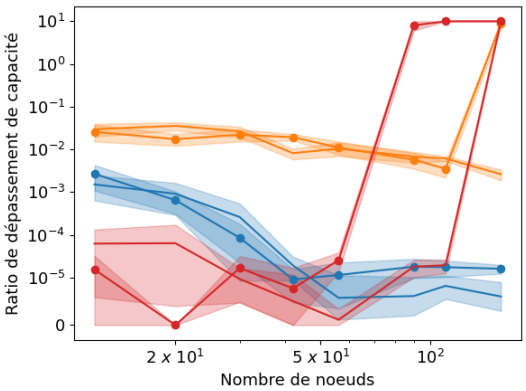
FIGURE 4.4 – Temps de calcul pour les algorithmes principaux sur tous les jeux de données



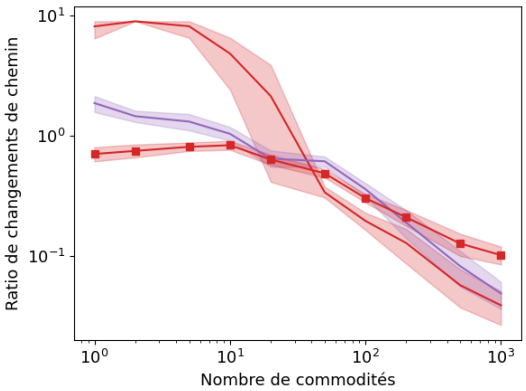
(a) Comparaison pénalisation, ratio de changements de chemin, jeu de données *grille facile*



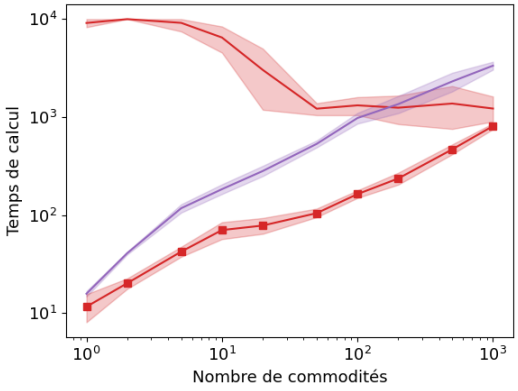
(b) Comparaison pénalisation, temps de calcul, jeu de données *grille facile*



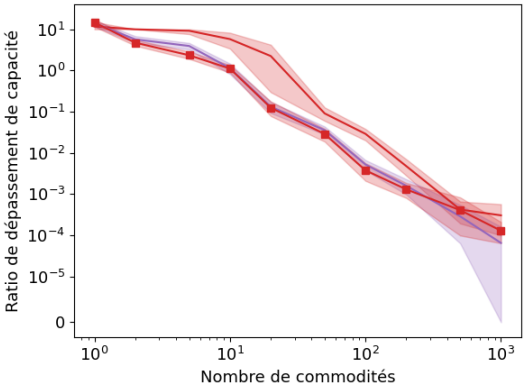
(c) Comparaison pénalisation, ratio de dépassement, jeu de données *grille facile*



(d) Ordre d'arrondi, ratio de changements de chemin, jeu de données *taille de commodités*



(e) Ordre d'arrondi, temps de calcul, jeu de données *taille de commodités*



(f) Ordre d'arrondi, ratio de dépassement, jeu de données *taille de commodités*

FIGURE 4.5 – Comparaison des algorithmes pénalisés et non pénalisés sur le jeu de données *grille facile* et comparaison de différents ordres d'arrondi sur le jeu de données *taille de commodités*

ratio de changements de chemin de la solution renvoyée (généralement de 2 à 4 fois) sauf quand la version non restreinte dépasse la limite des trois heures de temps de calcul. Quant au dépassement de capacité, la plupart du temps la différence n'est pas statistiquement significative. Globalement, la restriction de chemin semble introduire un compromis entre le temps de calcul et le nombre de changements de chemin.

Pénalisation du flot. Nous comparons les méthodes SRR-arc-nœud, SRR-chemin et SRR-séquence à leurs variantes sans pénalisation du flot dans les Figures 4.5a, 4.5b, 4.5c et dans l'annexe. Les versions non pénalisées donnent des temps de calcul plus longs, en particulier pour SRR-chemin où la version non pénalisée peut être jusqu'à 10 fois plus lente. En termes de ratio de changements de chemin, les versions pénalisées donnent de meilleurs résultats, surtout sur les instances ayant les plus grands graphes. La différence de dépassement de capacité des solutions n'est généralement pas statistiquement significative mais reste la plupart du temps en faveur des versions pénalisées. Globalement, la pénalisation a un impact positif sur tous les algorithmes testés, toutes les métriques et tous les jeux de données.

Ordre d'arrondi dans les méthodes multi pas de temps. Contrairement aux méthodes considérant un seul pas de temps pour lesquelles il existe un ordre préférable d'arrondi des variables (les commodités de plus grande demande en premier), ce n'est pas le cas pour les méthodes considérant plusieurs pas de temps. Nous comparons deux ordres d'arrondi différents dans SRR-séquence et SRR-séquence-pas-de-temps. Les résultats pour SRR-séquence-restreint sont donnés comme point de comparaison. Pour les ensembles de données où la taille du graphe varie, la plupart du temps, SRR-séquence donne des solutions ayant un meilleur ratio de changements de chemin. Son temps de calcul est plus petit sur les petites instances et similaire ou plus grand dans les grandes instances. Enfin, SRR-séquence renvoie généralement les solutions avec le plus mauvais taux de dépassement de capacité. Nous avons mis en évidence dans la Figure 4.5d, 4.5e et 4.5f les résultats pour le jeu de données où le nombre de commodités varie. Dans ce cas, SRR-séquence a des difficultés à résoudre les instances avec un petit nombre de commodités. C'est également le cas lorsque la demande maximale des commodités est approximativement égale aux capacités des arcs. Cependant, SRR-séquence tend à devenir meilleur que SRR-séquence-pas-de-temps lorsque le nombre de commodités est important.

Comparaison de la formulation arc-nœud agrégée et de la formulation par chemins. Deux formulations ont été utilisées pour calculer des relaxations linéaires exactes dans les méthodes considérant un seul pas de temps à la fois : la formulation par chemins dans SRR-chemin et la formulation arc-nœud agrégée dans SRR-arc-nœud. En termes de ratio de changements de chemin, les deux méthodes donnent des résultats assez similaires sur tous les jeux de données. En termes de temps de calcul, la plupart du temps, SRR-arc-nœud s'exécute plus rapidement que SRR-chemin. Notons que les deux méthodes exploitent le fait que nos instances ne contiennent qu'un faible nombre d'origines différentes. SRR-arc-nœud utilise des variables agrégées tandis que SRR-chemin fait moins d'appels à l'algorithme de plus court chemin de Dijkstra. Notons que le temps de calcul de SRR-arc-nœud peut devenir très grand lorsque le nombre de sources est important car cela implique un grand nombre de variables dans la formulation arc-nœud agrégée. Enfin, en termes de dépassement de capacité,

SRR-arc-nœud est plus performant que SRR-chemin lorsqu'il existe une différence statistique entre les deux algorithmes. Dans l'ensemble, l'utilisation d'une formulation arc-nœud agrégée donne de meilleurs résultats que l'utilisation d'une formulation par chemins.

Commentaires généraux. Bien que les méthodes prenant en compte plusieurs pas de temps soient significativement plus lentes que leurs homologues considérant un seul pas de temps, elles fournissent la plupart du temps des solutions de meilleure qualité. L'application d'une méthode *Branch and Bound* à la formulation par chemins donne de bons résultats lorsque l'ensemble des chemins utilisables est restreint. En effet, cette méthode réalise un bon compromis temps de calcul/ratio de changements de chemin et peut bénéficier de la présence de plusieurs processeurs même si de tels résultats ne sont pas présentés ici. Le fait qu'un chemin initial soit donné et préférable aux autres chemins semble aider les heuristiques internes des solveurs MILP commerciaux. En effet, de telles performances n'ont pas pu être reproduites sur le problème de flot insécable statique où aucun chemin n'est préféré parmi l'ensemble des chemins valides.

4.4 Conclusion

Dans ce chapitre, plusieurs nouvelles méthodes pour résoudre des instances de moyenne et grande taille du problème de flot insécable dynamique ont été présentées. En particulier, de nouvelles formulations ont été introduites qui modélisent soit le problème en nombres entiers, soit sa relaxation linéaire. De plus, nous avons introduit de nouvelles approches pour résoudre le problème de *pricing* utilisé pour résoudre la formulation introduite par Gamvros et Raghavan (2012). Ces méthodes ne reposent pas sur des calculs de k-plus-courts chemins et ne partagent donc pas les limitations de l'approche précédente : le nombre de k-plus-courts chemins à calculer peut être exponentiel en la taille du graphe considéré.

Les formulations ont été intégrées dans des solveurs matheuristiques qui ont été comparés sur plusieurs jeux de données. Ces solveurs viennent compléter l'approche de résolution exacte proposée par Gamvros et Raghavan (2012). Plusieurs aspects clés des solveurs expliquant leurs performances ont également été mis en évidence. Dans l'ensemble, ne considérer qu'un pas de temps à la fois et un petit ensemble de chemins autorisés pour chaque commodité donne un bon compromis entre la qualité de la solution et le temps de calcul lorsque la formulation obtenue est résolue à l'aide d'un solveur commercial. Cependant, les méthodes qui donnent les solutions avec la meilleure qualité reposent sur la formulation par séquence de chemins qui considère plusieurs pas de temps simultanément.

Bien que les métaheuristiques n'aient été étudiées ni dans ce travail ni dans la littérature sur le problème de flot insécable dynamique, elles pourraient constituer une alternative aux méthodes basées sur des formulations de programmation linéaire (en nombres entiers).

Comme nous venons de le voir au cours des deux derniers chapitres, la plupart des méthodes de résolution utilisées pour les problèmes de flot insécable statiques ou dynamiques reposent sur des calculs de relaxation linéaire. Ces méthodes pourraient donc utiliser une

relaxation linéaire plus forte pour obtenir de meilleurs résultats. C'est pourquoi dans le chapitre suivant, nous nous intéressons à des méthodes de décomposition capables d'améliorer la relaxation linéaire des problèmes de flot insécable. Nous y ferons une description de certaines méthodes connues de la littérature et proposerons notre propre méthode de décomposition.

Chapitre 5

Méthodes de décomposition pour le renforcement de relaxations linéaires

Résumé du chapitre

Dans ce chapitre, nous nous intéressons à des méthodes de décomposition capables d'améliorer la relaxation linéaire de programmes linéaires en nombres entiers et particulier du problème de flot insécable. Nous commençons une présentation des principaux résultats sur deux méthodes de décomposition connues de la littérature : les décompositions de Dantzig-Wolfe et de Fenchel. Nous présentons ensuite une nouvelle approche de résolution pour le sous-problème de Fenchel ainsi qu'une nouvelle méthode de décomposition. Cette méthode de décomposition utilise les problèmes maître de Dantzig-Wolfe et de Fenchel ainsi qu'un sous-problème de Fenchel. Nous la comparons expérimentalement aux méthodes de la littérature sur des instances du problème de flot insécable et obtenons des résultats très prometteurs.

La plupart des méthodes proposées dans cette thèse utilisent une relaxation linéaire afin de calculer une solution entière du problème de flot insécable. Dans ce chapitre, nous nous intéressons donc au renforcement de cette relaxation linéaire à l'aide de méthodes de décomposition. Ces méthodes sont appliquées aux contraintes de capacité du problème de flot insécable car ce sont les seules dont la relaxation linéaire peut être améliorée. En effet, il est largement admis que le polyèdre associé à des contraintes de flot est un polyèdre entier (tous ces sommets sont des points entiers). Sa relaxation linéaire ne peut donc pas être renforcée.

Le polyèdre associé à un contrainte de capacité du problème de flot insécable tel que formulé en Section 2.1 est le suivant :

$$\left\{ f_e^1, ..., f_e^K \in [0,1], o_e \in \mathbb{R}^+ \middle| \sum_{k \in K} f_e^k d^k \leq c_e + o_e \right\}.$$

Les sommets de ce polyèdre ne possèdent pas de propriété d'intégrité, c'est-à-dire qu'en

certain de ces sommets, certaines variables f_e^k prennent une valeur fractionnaire. C'est pourquoi l'application des méthodes de décomposition présentées ci-après à ces contraintes permet de renforcer la relaxation du problème de flot insécable. Des études ont été menées sur la structure, la recherche de coupe et le renforcement de la relaxation linéaire de ce type de polyèdre par Marchand et Wolsey (1999) ainsi que dans le cadre plus général des programmes linéaires en variables mixtes (Dash, 2011; Fukasawa et Goycoolea, 2011; Chvátal *et al.*, 2013). Par ailleurs, des méthodes d'optimisation sur ce polyèdre ont été étudiées par Büther et Briskorn (2012); Lin *et al.* (2011); Zhao et Li (2014); He *et al.* (2019); Liu (2017).

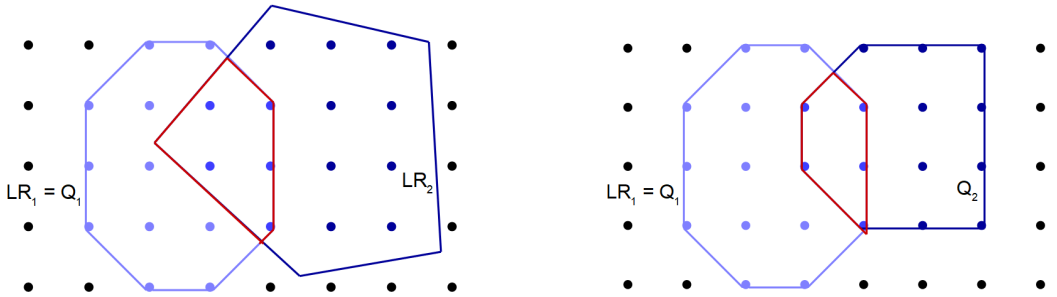
Afin de simplifier les notations et de ne rien perdre de la généralité des méthodes utilisées, nous présentons les travaux de ce chapitre à l'aide de notations générales. Par ailleurs, nous considérerons que les polyèdres étudiés sont bornés et ne possèdent donc pas de rayons extrêmes. Les méthodes de décomposition sont appliquées au modèle de programmation linéaire en nombres entiers suivant :

$$\begin{aligned}(P) \quad & \max_x c^T x \\ & \text{tel que} \\ & A_1 x \leq b_1 \\ & A_2 x \leq b_2 \\ & x \in X\end{aligned}$$

où X est un produit d'ensembles $\mathbb{R}$ et $\mathbb{Z}$. On considérera que les contraintes de flot du problème de flot insécable sont représentées par le bloc de contraintes $A_1 x \leq b_1$ tandis que les contraintes de capacité sont représentées par le bloc de contraintes $A_2 x \leq b_2$. Par ailleurs, on notera $\bar{X}$ la relaxation de l'ensemble X où les ensembles $\mathbb{Z}$ sont remplacés par des ensembles $\mathbb{R}$. Les polyèdres suivants sont considérés tout au long de ce travail :

$$\begin{aligned}LR_1 &= \{x \in \bar{X} | A_1 x \leq b_1\} \\ LR_2 &= \{x \in \bar{X} | A_2 x \leq b_2\} \\ Q_1 &= conv(\{x \in X | A_1 x \leq b_1\}) \\ Q_2 &= conv(\{x \in X | A_2 x \leq b_2\})\end{aligned}$$

Toutes les méthodes de décomposition présentées ci-après peuvent être utilisées pour renforcer la relaxation linéaire d'un problème de programmation linéaire en nombres entiers tel que (P). En effet, dans la relaxation linéaire traditionnelle du problème (P), l'espace de solution est $LR_1 \cap LR_2$ tandis qu'après les reformulations, le polyèdre LR_2 est remplacé par Q_2. L'espace de solution de la relaxation est alors $LR_1 \cap Q_2$ ce qui conduit à une relaxation plus forte puisque $Q_2 \subset LR_2$. Ce point est illustré en Figure 5.1.



(a) Polyèdres associés à la relaxation linéaire

(b) Polyèdres obtenus après les méthodes de décomposition

FIGURE 5.1 – Représentation des espaces de solution de la relaxation linéaire et des modèles issus des méthodes de décomposition

Dans la suite, toutes les méthodes de décomposition envisagées font l'hypothèse qu'il existe un algorithme efficace pour optimiser une fonction linéaire sur le polyèdre Q_2. Cet algorithme est appelé oracle d'optimisation et résout le programme linéaire en nombres entiers suivant :

$$
\begin{aligned}
(O) \quad & \max_{x} \pi x \\
& \text{tel que} \\
& A_2 x \leq b_2 \\
& x \in X
\end{aligned}
$$

où π est le gradient de la fonction linéaire à optimiser. Dans le cas des flots insécables, le polyèdre Q_2 est un polyèdre de sac à dos associé aux contraintes de capacité. L'oracle (O) est donc un algorithme résolvant efficacement le problème du sac à dos. Notons que l'on peut aussi définir un problème d'optimisation sur le polyèdre associé aux contraintes de flot et que celui-ci peut être résolu avec un algorithme de plus court chemin.

Dans la suite de ce chapitre, nous faisons en Section 5.1, 5.2 et 5.3 une présentation de deux méthodes de décomposition connues de la littérature : les décompositions de Dantzig-Wolfe et de Fenchel. Cette présentation ne se veut pas exhaustive mais introduit tous les éléments nécessaires à la compréhension de la nouvelle méthode de décomposition que nous présentons en Section 5.5. Cette méthode utilise les problèmes maître de Dantzig-Wolfe et de Fenchel ainsi qu'un sous-problème de Fenchel. De plus, nous présentons en Section 5.4 une nouvelle approche de résolution pour le sous-problème de Fenchel. Enfin, en Section 5.6 l'ensemble de ces méthodes est appliqué au problème de flot insécable et évalué expérimentalement sur des instances de petite et moyenne taille de ce problème. Dans ce contexte, la nouvelle décomposition proposée démontre de bon résultats.

5.1 La décomposition de Dantzig-Wolfe

La décomposition de Dantzig-Wolfe est l'une des méthodes de décomposition les plus étudiées de la littérature au côté des décompositions Lagrangienne et de Benders. Son champ d'application est vaste car elle permet à la fois d'accélérer la résolution des programmes linéaires de grandes tailles et d'améliorer la relaxation linéaire de la plupart des modèles de programmation linéaire en nombres entiers. Elle a par ailleurs connu de nombreux succès sur des cas d'application variés ce qui explique l'attention qu'elle a reçue au fil des années (Desaulniers *et al.*, 2006).

5.1.1 Présentation de la méthode

L'idée principale lors de l'application de la décomposition de Dantzig-Wolfe au problème (P) est de changer la manière d'imposer la condition $x \in Q_2 = conv(\{x \in X | A_2 x \leq b_2\})$. Au lieu, d'utiliser les inégalités $A_2 x \leq b_2$, le théorème de Minkowski-Weyl (Wolsey et Nemhauser, 1999) indique qu'il est possible d'assurer cette condition en réécrivant les variables x comme une combinaison convexe de sommets de Q_2. Pour cela, une nouvelle variable λ_i est introduite pour chaque sommet x_i de Q_2. Elle représente le poids de ce sommet dans la combinaison convexe. Cette transformation conduit à la formulation suivante :

$$
\begin{aligned}
(DW) \quad & \max_{x,\lambda_i} c^T x \\
& \text{tel que} \\
& A_1 x \leq b_1 \\
& \sum_{i \in I} \lambda_i x_i = x \\
& \sum_{i \in I} \lambda_i = 1 \\
& x \in X \\
& \lambda_i \in \mathbb{R}^+, \quad \forall i \in I
\end{aligned}
$$

Puisqu'il existe une variable λ_i pour chaque sommet du polyèdre Q_2, leur nombre est trop grand pour qu'elles soient toutes explicitement prises en compte dans le problème (DW). Il est nécessaire d'en considérer la plupart implicitement et de les générer progressivement à l'aide d'une méthode de génération de colonnes. Dans une génération de colonnes, à chaque itération, la variable améliorant localement le plus la fonction objectif est ajoutée dans le problème. Ceci s'effectue en calculant, à l'aide d'un sous-problème d'optimisation dit de *pricing*, la variable ayant un coût réduit maximum. Pour plus de détails sur la génération de colonnes, se référer à l'Annexe A.1. Dans le problème (DW), le coût réduit d'une variable λ_i associée à un sommet x_i de Q_2 s'exprime en fonction des variables duales associées aux contraintes $\sum_{i \in I} \lambda_i x_i = x$ et $\sum_{i \in I} \lambda_i = 1$, notées respectivement π et π_0. La formule du coût réduit est alors $\pi x_i + \pi_0$. On

constate que la partie du coût réduit π_0 est la même pour tous les variables λ_i et n'impacte donc pas laquelle de ces variables a le plus grand coût réduit. Trouver la variable de coût réduit maximum revient donc à trouver le sommet x de Q_2 maximisant la fonction linéaire πx ce qui est exactement un appel à l'oracle d'optimisation (O). Une fois que suffisamment de variables ont été ajoutées au problème principal (DW), la variable de coût réduit maximum a un coût réduit négatif ce qui indique que la valeur de la fonction objectif ne peut plus être améliorée. La solution du problème principal est alors optimale et la procédure s'arrête.

5.1.2 Limites de la méthode et techniques de stabilisation

La décomposition de Dantzig-Wolfe présentée ci-dessus est une technique utilisée dans une large gamme d'applications mais qui souffre de nombreux défauts qui ralentissent sa convergence. En particulier, on peut mentionner les limites suivantes (Pessoa *et al.*, 2013) :

— **Oscillations duales :** au cours de l'algorithme les valeurs des variables duales π, π_0 utilisées pour générer les sommets de Q_2 effectuent de grandes oscillations. De plus, d'une itération à la suivante, les variables duales ne se rapprochent pas forcément de leur valeur optimale ce qui ralentit leur convergence.

— **L'effet de ralentissement :** vers la fin de l'algorithme, la taille de l'espace des solutions duales se réduit marginalement à chaque itération, ce qui rend la convergence des variables duales vers leur valeur optimale très lente.

— **Dégénérescence du primal et solutions duales équivalentes :** Le problème principal (DW) est régulièrement dégénéré. En particulier, à certaines itérations, plusieurs valeurs pour les variables duales peuvent être optimales. Dans ce cas, la méthode itère entre des solutions duales équivalentes sans faire de progrès sur la valeur de la fonction objectif ce qui ralentit la convergence.

Afin de pallier ces difficultés, des méthodes de stabilisation des variables duales ont été envisagées. Ces méthodes peuvent être classifiées en trois catégories (Pessoa *et al.*, 2013) :

— **Pénalisation :** les méthodes de pénalisation s'interprètent mieux en considérant le dual du problème (DW). Afin de stabiliser les variables duales π une pénalisation $f(\|\pi - \hat{\pi}\|)$ est ajoutée à la fonction objectif du dual. Dans cette pénalisation, $f()$ est une fonction croissante, ce qui pousse π à rester proche d'une valeur $\hat{\pi}$ qui évolue doucement au cours de l'algorithme. Typiquement, $\hat{\pi}$ est une des valeurs prises par les variables duales lors des itérations précédentes. Un cas particulier largement étudié est de pénaliser proportionnellement à $\|\pi - \hat{\pi}\|_2^2$, ce qui est fait dans les méthodes de *Bundle* (Briant *et al.*, 2008).

— **Lissage :** dans les méthodes de lissage, les variables π du dual du problème (DW) ne sont pas utilisées directement dans le sous-problème afin de générer de nouveaux sommets de Q_2. Notons, pour l'itération j de la génération de colonnes, π_j les valeurs des variables issues du dual du problème (DW) et $\hat{\pi}_j$ les valeurs utilisées dans le sous-problème. Une méthode de lissage proposée par Neame (2000) utilise la formule suivante : $\hat{\pi}_j = \alpha\hat{\pi}_{j-1} + (1-\alpha)\pi_j$. Cette méthode revient à ajouter un effet de *momentum* aux variables duales. Une autre méthode proposée par Wentges (1997), effectue une combinaison convexe avec une valeur duale fixe $\hat{\pi}$, c'est à dire $\hat{\pi}_j = \alpha\hat{\pi} + (1-\alpha)\pi_j$.

— **Centralisation :** l'idée des méthodes de centralisation est qu'il est plus efficace d'utiliser dans le sous-problème des valeurs duales situées à l'intérieur du polyèdre dual plutôt que sur un sommet extrême du polyèdre dual. De tels points intérieurs sont en revanche plus coûteux à calculer que des points extrêmes. Le point intérieur utilisé dans la Génération de Colonne Primale-Duale (Gondzio *et al.*, 2013) est celui obtenu en résolvant approximativement le problème (DW) par une méthode de points intérieurs. Un autre point classique est le centre analytique utilisé dans la méthode de plan sécant du centre analytique (Goffin et Vial, 2002).

5.2 La décomposition de Fenchel

La décomposition de Fenchel est une méthode proposée par Boyd (1994) et destinée principalement à améliorer la relaxation des programmes linéaires en nombres entiers. La décomposition de Fenchel est une méthode de génération de coupes alimentée par un algorithme de séparation exacte des polyèdres. Cette technique a rencontré un certain succès dans plusieurs applications telles que des problèmes de sac à dos (Boyd, 1993; Kaparis et Letchford, 2010), des problèmes d'affectation généralisée (Avella *et al.*, 2010), des problèmes de *network design* avec flot insécable (Chen *et al.*, 2021) ou encore des problèmes d'optimisation stochastique (Ntaimo, 2013; Beier *et al.*, 2015).

5.2.1 Description de la méthode

La décomposition de Fenchel est une méthode de plans sécants utilisant un algorithme de séparation exacte pour générer des inégalités violées par la solution de la relaxation linéaire d'un problème. Pour fonctionner, l'algorithme de séparation d'un polyèdre tel que Q_2 suppose qu'il existe un oracle tel que l'oracle (O), c'est à dire capable d'optimiser une fonction linéaire sur Q_2. Nous détaillons maintenant comment appliquer la décomposition de Fenchel au problème (P).

Tout d'abord, un problème maître est créé à partir de la relaxation linéaire du problème (P). Celui-ci contient un petit sous-ensemble $\mathcal{C}$ des inégalités valides pour le polyèdre Q_2, dans la plupart des cas initialement égal à $A_2x \leq b_2$. Le problème maître s'exprime alors de la façon suivante :

$$
\begin{aligned}
(F) \quad & \max_x c^T x \\
& \text{tel que} \\
& A_1 x \leq b_1 \\
& \pi x \leq \pi_0 \quad \forall (\pi, \pi_0) \in \mathcal{C} \subset \mathcal{C}(Q_2) \\
& x \in \bar{X}
\end{aligned}
$$

où $\mathcal{C}(Q_2)$ est l'ensemble des coupes valides pour le polyèdre Q_2. Des inégalités de $\mathcal{C}(Q_2)$ sont ensuite ajoutées progressivement via un algorithme de plans sécants jusqu'à ce que la solution du problème maître appartienne à Q_2. Pour ce faire, à chaque fois qu'une solution $\hat{x}$ du problème maître est calculée, une coupe séparant $\hat{x}$ du polyèdre Q_2 est générée. Si une telle coupe existe, elle est ajoutée au problème maître qui calcule une nouvelle solution $\hat{x}$. En revanche, s'il n'existe pas de coupe séparant $\hat{x}$ du polyèdre Q_2 alors $\hat{x}$ appartient à Q_2 et la procédure s'arrête.

5.2.2 Résolution du problème de séparation

La génération d'une coupe $\pi x \leq \pi_0$ séparant le polyèdre Q_2 d'une solution $\hat{x}$ du problème maître (F) est effectuée de manière exacte par le sous-problème de séparation suivant :

$$
\begin{aligned}
(S) \quad & \max_{\pi,\pi_0} \pi\hat{x} - \pi_0 \\
& \text{tel que} \\
& \pi x_i \leq \pi_0, \quad \forall x_i \in Q_2 \\
& \pi, \pi_0 \in \mathbb{R}
\end{aligned}
$$

Dans le sous-problème de séparation (S), l'objectif maximise la violation par $\hat{x}$ de la coupe générée tandis que les contraintes garantissent que la coupe est valide pour le polyèdre Q_2. Par conséquent, le point $\hat{x}$ peut être séparé de Q_2 si et seulement si le problème (S) admet une solution de valeur positive. Cependant, notons que si une coupe $\pi x \leq \pi_0$ est valide pour tous les points de Q_2 alors il en est de même de la coupe $\alpha\pi x \leq \alpha\pi_0$ pour une constante α positive quelconque. L'espace des solutions du problème (S) est donc un cône ce qui empêche de résoudre ce problème tel quel. En effet, lorsqu'une solution de valeur strictement positive existe, cette solution correspond à un rayon non borné du problème (S). La valeur optimale de ce problème est alors $+\infty$. Par ailleurs, le point $(\pi, \pi_0) = (0, 0)$ est toujours une solution valide du problème (S). Afin de rendre ce problème borné, un processus de normalisation, détaillé en Section 5.3, doit être effectué. Cette normalisation impacte grandement la coupe générée (qui peut être une facette de Q_2 ou non) ainsi que les propriétés de convergence de la méthode de décomposition.

Le problème de séparation (S) possède une contrainte pour chaque sommet de Q_2. De par leur grand nombre, ces contraintes doivent être considérées implicitement et ajoutées via un algorithme de génération de contraintes. Pour cela, le problème (S) est résolu avec un sous-ensemble de ses contraintes et retourne une solution $(\hat{\pi}, \hat{\pi_0})$. Vérifier si l'une des contraintes de problème (S) est violée peut être effectué en trouvant le sommet de Q_2 qui viole le plus la coupe $\hat{\pi}x \leq \hat{\pi_0}$. Ceci peut être effectué en faisant un appel à l'oracle d'optimisation (O) pour la fonction linéaire associée à $\hat{\pi}$. Si la valeur optimale de l'oracle est supérieure à $\hat{\pi_0}$

alors la solution x^* de l'oracle est un sommet de Q_2 violant la coupe $\hat{\pi}x \leq \hat{\pi_0}$. Une contrainte $\pi x^* \leq \pi_0$ est alors rajoutée au problème (S) et celui-ci calcule de nouveaux coefficients $(\hat{\pi}, \hat{\pi_0}$. En revanche, si la valeur optimale de l'oracle est inférieure à $\hat{\pi_0}$ et donc que x^* vérifie la coupe, alors par construction, tous les points de Q_2 vérifient la coupe $\hat{\pi}x \leq \hat{\pi_0})$ qui est donc une solution optimale du problème (S).

5.2.3 Démarche équivalente : combinaison des décompositions de Dantzig-Wolfe et de Benders

Dans cette section, nous montrons que l'application d'une décomposition de Benders à un problème reformulé avec une décomposition de Dantzig-Wolfe conduit à la formulation obtenue après une décomposition de Fenchel. Cette équivalence est aussi décrite dans le livre de Martin (2012) dans la partie 5.1 du Chapitre 16.

On considère la reformulation (DW) de la Section 5.1.1 du problème (P). La première étape d'une décomposition de Benders consiste à placer certaines variables dans un sous-problème séparé. En appliquant cette opération aux variables λ du problème (DW), le problème principal et le sous-problème suivants sont obtenus :

$$
\begin{aligned}
(F2) \quad & \max_x c^T x + z \\
& \text{tel que} \\
& A_1 x \leq b_1 \\
& z \leq F(x) \\
& x \in X
\end{aligned}
$$

$$
\begin{aligned}
(D) \quad F(\hat{x}) = & \max_{\lambda_i} 0 \\
& \text{tel que} \\
& \sum_{i \in I} \lambda_i x_i = \hat{x} \\
& \sum_{i \in I} \lambda_i = 1 \\
& \lambda_i \in \mathbb{R}^+, \quad \forall i \in I
\end{aligned}
$$

Le problème (D) effectue la décomposition de $\hat{x}$ en une combinaison convexe de sommets de Q_2. Ce sous-problème renvoie toujours une valeur de zéro à moins qu'il ne soit infaisable. Cette infaisabilité se produit si et seulement si le point $\hat{x}$ n'appartient pas au polyèdre Q_2 et

dans ce cas, par convention, $F(\hat{x}) = +\infty$. La seconde étape d'une décomposition de Benders consiste à remplacer le sous-problème par son dual. Si les variables duales des contraintes $\sum_{i\in I} \lambda_i x_i = x$ et $\sum_{i\in I} \lambda_i = 1$ sont notées respectivement $-\pi$ et π_0 et que l'on exprime le dual avec une maximisation alors le dual du problème (D) est le problème de séparation (S) obtenu dans la décomposition de Fenchel.

La procédure de décomposition de Benders distingue deux cas lors de la résolution du problème dual (S) : soit (S) admet une solution optimale, soit il est non borné. Comme expliqué en Section 5.2.2, le problème (S) admet $(\pi, \pi_0) = (0, 0)$ comme solution optimale lorsque $\hat{x} \in Q_2$ et est non borné dans le cas contraire. Lorsque le sous-problème (S) est non borné dans une direction $(\hat{\pi}, \hat{\pi_0})$ la décomposition de Benders indique qu'une contrainte $\hat{\pi}x \leq \hat{\pi_0}$ doit être ajoutée au problème principal. Comme dans l'approche précédente, afin d'éviter que le sous-problème (S) ne soit non borné, un processus de normalisation est effectué. Dans le deuxième cas, lorsque le sous-problème (S) admet une solution optimale $(\hat{\pi}, \hat{\pi_0})$, la décomposition de Benders indique qu'une contrainte $z \leq \hat{\pi_0} - \hat{\pi}x$ doit être ajoutée au problème principal. Or, le problème (S) admet uniquement $(0, 0)$ comme solution optimale car $(0, 0)$ est l'unique sommet de son espace de solution. La contrainte alors ajoutée est $z \leq 0$. Par conséquent, la variable z prendra toujours la valeur 0 ce qui implique que la variable z et la contrainte $F(x) \leq z$ peuvent être supprimées du problème maître.

Une fois que la variable z et la contrainte $F(x) \leq z$ ont été écartées, on obtient le même problème maître et le même sous-problème de séparation (S) que dans la décomposition de Fenchel. Ceci achève de prouver l'équivalence entre les deux démarches.

5.2.4 Pré-traitements et post-traitements

Coupes triviales et heuristiques : La résolution du problème de séparation exacte (S) est une procédure assez coûteuse. Il est souvent intéressant en terme de temps de calcul de vérifier s'il n'existe pas une méthode plus rapide pour calculer une coupe. En particulier, deux types de méthodes sont souvent utilisés. Le premier consiste à voir dans certains cas très simples s'il existe une coupe triviale (par exemple si le problème est en très faible dimension, voir Réduction de la dimensionnalité). Ce type de méthodes est présenté pour les polyèdres de sac à dos par Vasilyev *et al.* (2016) et de *network desgin* par Chen *et al.* (2021). Le deuxième type de méthodes consiste à être capable de créer des coupes de manière heuristique. Par exemple, il existe un très grand nombre de procédures pour créer des coupes pour les polyèdres de de sac à dos, telles que les inégalités de *Mixed Integer Rounding* ou les *lifted cover cuts* pour n'en citer que deux. Ces méthodes heuristiques ne génèrent pas toujours des facettes du polyèdre à séparer mais peuvent être utilisées avant d'appliquer la procédure de séparation exacte afin de gagner du temps de calcul.

Réduction de la dimensionnalité : Une technique présentée par Boccia *et al.* (2008) et permettant une large diminution du temps de résolution du problème de séparation (S) est la réduction de la dimensionnalité de ce problème. Cette technique s'applique lorsque les variables x du problème initial (P) possèdent des bornes qui sont régulièrement atteintes. Le

cas le plus répandu et le plus simple étant celui des variables binaires, nous faisons l'exposition de cette méthode dans ce cadre. Au lieu de séparer un point $\hat{x}$ de Q_2, ce point est séparé du sous-polyèdre $Q_2^f(\hat{x})$ induit par les variables prenant une valeur différente de leurs bornes dans $\hat{x}$ (dans le cas binaire, induit par les variables prenant une valeur fractionnaire dans $\hat{x}$). Plus précisément, $Q_2^f(\hat{x}) = \{x \in Q_2 | x_i = \hat{x}_i \text{ si } \hat{x}_i \in \{0,1\}\}$. En effet, $\hat{x}$ appartient à Q_2 si et seulement si il appartient à $Q_2^f(\hat{x})$ ce qui implique que $\hat{x}$ est séparable de Q_2 si et seulement si il est séparable de $Q_2^f(\hat{x})$. Remplacer Q_2 par $Q_2^f(\hat{x})$ dans le problème de séparation présente deux avantages. Premièrement, le problème ne considère que les variables prenant une valeur fractionnaire dans $\hat{x}$ ce qui réduit le nombre de dimensions du problème et accélère donc sa résolution. Forcer certaines variables à prendre une valeur binaire dans l'oracle d'optimisation sur Q_2 est d'ailleurs souvent facile à effectuer. Deuxièmement, si le problème de séparation (S) est résolu directement par le biais d'une génération de contraintes, une majeure partie du temps de calcul est utilisée pour générer des contraintes associées à des sommets de Q_2 n'appartenant pas à $Q_2^f(\hat{x})$ et qui sont donc pour la plupart inutiles. Réduire l'espace de recherche à $Q_2^f(\hat{x})$ permet de concentrer le calcul sur la partie de l'espace contenant les sommets les plus importants.

Lifting **:** Lors de l'utilisation de la technique de "Réduction de dimensionnalité précédente", la coupe générée est valide pour $Q_2^f(\hat{x})$ mais pas forcément valide pour Q_2. Une procédure permettant de créer une coupe valide pour Q_2 à partir de celle valide pour $Q_2^f(\hat{x})$ est la procédure de *lifting* séquentiel (Wolsey et Nemhauser, 1999). Dans la suite, nous nommerons variables fixées, les variables éliminées du problème de séparation lors du remplacement du polyèdre Q_2 par le polyèdre $Q_2^f(\hat{x})$. Pour les variables fixées, les coefficients de la coupe générée sont tous nuls. A chaque étape de la procédure, une nouvelle valeur pour le coefficient de l'une des variables fixées est calculée de telle sorte à ce que la coupe devienne valide pour le polyèdre où cette variable n'est plus fixée. Ces nouveaux coefficients sont optimaux dans le sens où si la coupe initiale était une facette du polyèdre $Q_2^f(\hat{x})$ alors la coupe issue du *lifting* séquentiel est une facette de Q_2. Nous présentons maintenant une itération de la procédure de *lifting* séquentiel. Supposons pour cela que $Q_2^f(\hat{x}) = \{x \in Q_2 | x_1 = 1\}$; la procédure ne contiendra alors qu'une seule itération car seule la variable x_1 est fixée. La coupe générée pour $Q_2^f(\hat{x})$ est $\pi x \leq \pi_0$ où le coefficient π_1, le coefficient associé à la variable x_1, est nul. La procédure de *lifting* consiste à créer une nouvelle coupe $\gamma x \leq \gamma_0$ telle que les deux coupes soient identiques lorsque $x_1 = 1$ et telle que la nouvelle coupe soit valide pour Q_2. Puisque $\pi x \leq \pi_0$ est déjà valide pour $Q_2^f(\hat{x})$, les deux conditions précédentes s'écrivent :

$$\forall x \in \{x | x_1 = 1\},\ \gamma x - \gamma_0 = \pi x - \pi_0$$
$$\max_{x \in \{x \in Q_2 | x_1 = 0\}} \gamma x \leq \gamma_0$$

Soit e_i, le $i^{\text{ème}}$ vecteur de la base canonique. En appliquant la première condition pour $x = e_1$, on obtient $\gamma_1 - \gamma_0 = \pi_1 - \pi_0$. De plus, en l'appliquant en $x = e_1 + e_i$ pour tout $i > 1$, on obtient $\gamma_1 + \gamma_i - \gamma_0 = \pi_1 + \pi_i - \pi_0$ ce qui devient $\gamma_i = \pi_i$ en soustrayant l'égalité précédente.

On sait donc que :

$$\forall i > 1, \gamma_i = \pi_i$$
$$\gamma_1 = \gamma_0 + \pi_1 - \pi_0$$

Il ne manque donc plus que de connaître la valeur de γ_0 pour connaître la valeur de tous les γ_i. En remplaçant γ_i par π_i pour tout $i > 1$ dans la deuxième condition on obtient $\gamma_0 \geq \max_{x \in \{x \in Q_2 | x_1 = 0\}} \pi x$. La plus petite valeur de γ_0 vérifiant la deuxième condition est donc $\gamma_0 = \max_{x \in \{x \in Q_2 | x_1 = 0\}} \pi x$ ce qui peut être calculé avec un appel à l'oracle d'optimisation sur Q_2 où la variable x_1 fixé à 0. Il est donc possible de calculer chaque coefficient *lifté* à l'aide d'un unique appel à l'oracle (O).

Erreurs numériques : Des erreurs d'arrondi peuvent se produire lors de la résolution du problème de séparation (S) et lors des appels à l'oracle d'optimisation, surtout si le vecteur π donné en entrée de l'oracle n'est pas constitué de nombres entiers. De telles erreurs peuvent conduire à des coupes faibles ou même invalides. Afin de se prémunir contre ces erreurs numériques, les coefficients π, π_0 de la coupe générée sont mis à l'échelle pour les rendre tous entiers. Ceci peut être effectué à l'aide du programme linéaire en nombres entiers suivant introduit par Boccia *et al.* (2008) :

$$\min_{t,\pi,\pi_0} t$$
$$\text{tel que}$$
$$\pi = t\hat{\pi}$$
$$\pi_0 = t\hat{\pi_0}$$
$$t \geq 1,\ \pi, \pi_0 \in \mathbb{Z}$$

Une approche plus heuristique consiste à tester tous les paramètres de mise à l'échelle inférieurs à 10^4 comme proposé par Vasilyev *et al.* (2016). La dernière étape consiste à vérifier que la coupe mise à l'échelle est toujours valide pour le polyèdre Q_2. Ceci peut être vérifié à l'aide d'un appel à l'oracle d'optimisation sur Q_2.

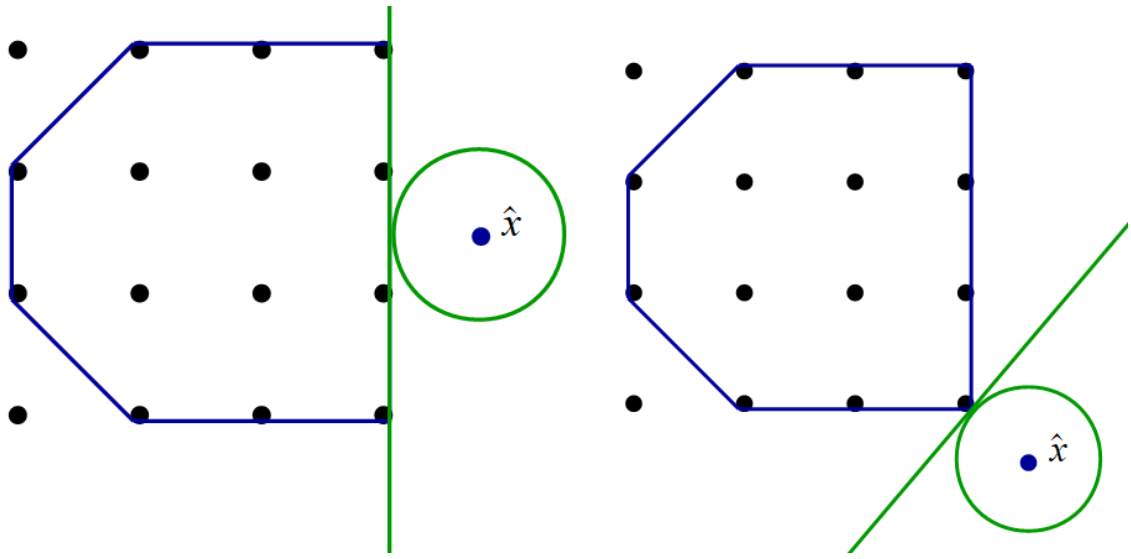
5.3 Démarches de normalisation dans la décomposition de Fenchel

Lorsque le point $\hat{x}$ à séparer de Q_2 n'appartient pas à Q_2, le problème de séparation (S) de la décomposition de Fenchel est non borné. Afin de le rendre borné, les coefficients π et π_0 de la coupe générée doivent être normalisés. Le choix de cette normalisation impacte fortement quelle coupe est maximalement violée par $\hat{x}$. Par ailleurs, en fonction de la normalisation

choisie, la coupe obtenue ne correspond pas forcément à une facette du polyèdre Q_2. L'utilisation d'une normalisation dans le problème (S) impacte aussi son problème dual (D). Il est souvent plus intuitif d'interpréter l'impact d'une normalisation dans le dual et nous utilisons donc cette approche pour les interprétations géométriques.

5.3.1 Normalisation $\|\pi\| \leq 1$

Dans ses travaux, Boyd (1995) analyse la normalisation $\|\pi\| \leq 1$ pour une norme quelconque. Avec ce choix, on peut considérer que la quantité optimisée est $\frac{\pi\hat{x}-\pi_0}{\|\pi\|}$, c'est à dire la distance de $\hat{x}$ à l'hyperplan $\pi x = \pi_0$. Notons que cette distance à l'hyperplan n'est pas mesurée en norme $\|.\|$ mais en sa norme duale $\|.\|^*$: $\|\lambda\|^* = \max_{\|x\|\leq 1} \lambda x$. En particulier, notons que les normes $\|.\|_1$ et $\|.\|_\infty$ sont duales et que la norme $\|.\|_2$ est auto-duale. Un des résultats importants du travail de Boyd (1995) est que l'enveloppe convexe de Q_2 peut être calculée en temps fini avec cette normalisation. En revanche, les coupes générées ne sont pas toujours des facettes du polyèdre Q_2. Avec cette normalisation, le problème dual (D) devient : trouver le point $x \in Q_2$ minimisant la distance entre x et $\hat{x}$ au sens de la norme duale. Ce point se situe sur la frontière de Q_2 et correspond au point de contact entre Q_2 et la plus petite sphère centrée en $\hat{x}$ touchant Q_2. La coupe générée est alors une coupe tangente à Q_2 et à la sphère au point de contact. Toutes ces interprétations géométriques sont illustrées en norme $\|.\|_2$ dans la Figure 5.2 où l'on peut voir que la coupe générée n'est pas toujours une facette de Q_2.



(a) Coupe associée à une normalisation en norme $\|.\|_2$: une facette est générée

(b) Coupe associée à une normalisation en norme $\|.\|_2$: la coupe générée n'est pas une facette

FIGURE 5.2 – Exemple de coupes générées avec une normalisation en norme $\|.\|_2$

5.3.2 Normalisations garantissant la génération de facettes

Nous nous intéressons maintenant à des normalisations permettant d'obtenir des facettes de Q_2. Pour ce faire, nous commençons par présenter un théorème faisant le lien entre les facettes de Q_2 et les rayons extrêmes du cône des solutions du problème de séparation (S). Ce théorème peut être trouvé dans (Conforti et Wolsey, 2019) (Proposition 1).

Théorème 5.1
Soit P un polyèdre non vide, soit $(\pi^j x = \pi_0^j)_{j\in J^=}$ une représentation non redondante de l'enveloppe affine de P et soit $(\pi^j x \leq \pi_0^j)_{j\in J^\leq}$ l'ensemble des facettes de P. Alors l'ensemble des coupes valides pour P est :

$$\mathcal{C}(P) = cone\left(\begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}\pi^j\\ \pi_0^j\end{pmatrix}_{j\in J^\leq}\right) + vect\left(\begin{pmatrix}\pi^j\\ \pi_0^j\end{pmatrix}_{j\in J^=}\right)$$

De plus, tous ces vecteurs sont nécessaires dans la représentation précédente.

Ce théorème affirme qu'à part le rayon trivial $(0, 1)$, les rayons extrêmes du cône des solutions de (S) sont associés aux facettes du polyèdre Q_2. Lorsque le polyèdre n'est pas de dimension pleine (*i.e.* $J^= \neq \emptyset$), les coupes correspondant à $(\pi^j, \pi_0^j)_{j\in J^=}$ sont les faces impropres de Q_2. Supposons pour la suite que Q_2 est de dimension pleine. On peut se ramener à ce cas une fois que toutes les faces impropres ont été générées.

Une condition suffisante assurant que les coupes générées soient des facettes de Q_2 est d'utiliser une condition de normalisation qui puisse être appliquée en ajoutant une unique contrainte linéaire au problème de séparation. Si une seule contrainte linéaire est ajoutée à (S), alors tous les points extrêmes du nouvel espace de solution correspondent à d'anciens rayons extrêmes et donc à des facettes de Q_2. Si le problème (S) est, en effet, rendu borné par l'ajout de la contrainte linéaire, alors il suffit de trouver un sommet optimal pour garantir la construction d'une facette. Plus généralement, il est possible d'ajouter plusieurs contraintes, tant qu'elles ne se croisent pas à l'intérieur du cône de solution de (S). Par exemple, l'ajout d'une contrainte $-1 \leq f(\pi, \pi_0) \leq 1$, où f est une fonction linéaire, correspond à l'ajout de deux contraintes linéaires qui ne se croisent pas. Les coupes générées induiront alors des facettes de Q_2. Lorsque, pour des raisons spécifiques au problème, nous savons que les coupes valides vérifient $\pi \geq 0$ (par exemple si Q_2 est un polyèdre de sac à dos), alors la condition $\|\pi\|_1 \leq 1$ se réduit à $\sum_i \pi_i \leq 1$ et peut être utilisée pour générer des inégalités induisant des facettes.

Normalisation de π_0 : Une normalisation garantissant la génération de facettes est $|\pi_0| \leq 1$. Cette normalisation est étudiée en détail par Conforti et Wolsey (2019). Elle rend borné le problème (S) si et seulement si $\hat{x}$ peut s'écrire comme une combinaison linéaire à coefficients positifs d'éléments de Q_2. En effet, regardons l'impact de la normalisation sur le

problème dual (D) qui devient :

$$\begin{aligned}(D) \quad & \min_{\lambda_i,\ z} |z| \\ & \text{tel que} \\ & \sum_{i\in I} \lambda_i x_i = \hat{x} \\ & \sum_{i\in I} \lambda_i = 1 + z \\ & z \in \mathbb{R},\ \lambda_i \in \mathbb{R}^+, \quad \forall i \in I\end{aligned}$$

En langage courant le problème dual (D) s'écrit donc : trouver une combinaison linéaire à coefficients positifs d'éléments de Q_2 égale à $\hat{x}$ dont la somme des coefficients est la plus proche possible de 1. Le problème de séparation (S) est borné si et seulement si son dual (D) est faisable. Or, le problème (D) est faisable si et seulement si $\hat{x}$ peut s'écrire comme une combinaison linéaire à coefficients positifs d'éléments de Q_2. Cette condition est naturellement atteinte lorsque l'origine se trouve dans l'intérieur de Q_2 puisque l'ensemble des combinaisons à coefficients positifs d'éléments de Q_2 est alors l'espace tout entier. Si l'on connaît un point dans l'intérieur de Q_2, il est possible d'assurer cette condition en translatant le problème pour placer l'origine sur ce point intérieur. Lorsque l'origine se situe dans l'intérieur de Q_2, la coupe générée est une facette de Q_2 contenant le point d'intersection entre la frontière de Q_2 et le segment reliant $\hat{x}$ et l'origine.

Normalisation directionelle de π : Une deuxième normalisation, étudiée par Bonami (2003) et garantissant la génération de coupes, consiste à borner les coefficients de π dans une direction donnée en utilisant la contrainte $|(\bar{x} - \hat{x})\pi| \leq 1$ où $\bar{x}$ est un point quelconque de l'espace. Afin de savoir quand cette normalisation rend le problème (S) borné, regardons son impact sur le problème (D). Celui-ci devient :

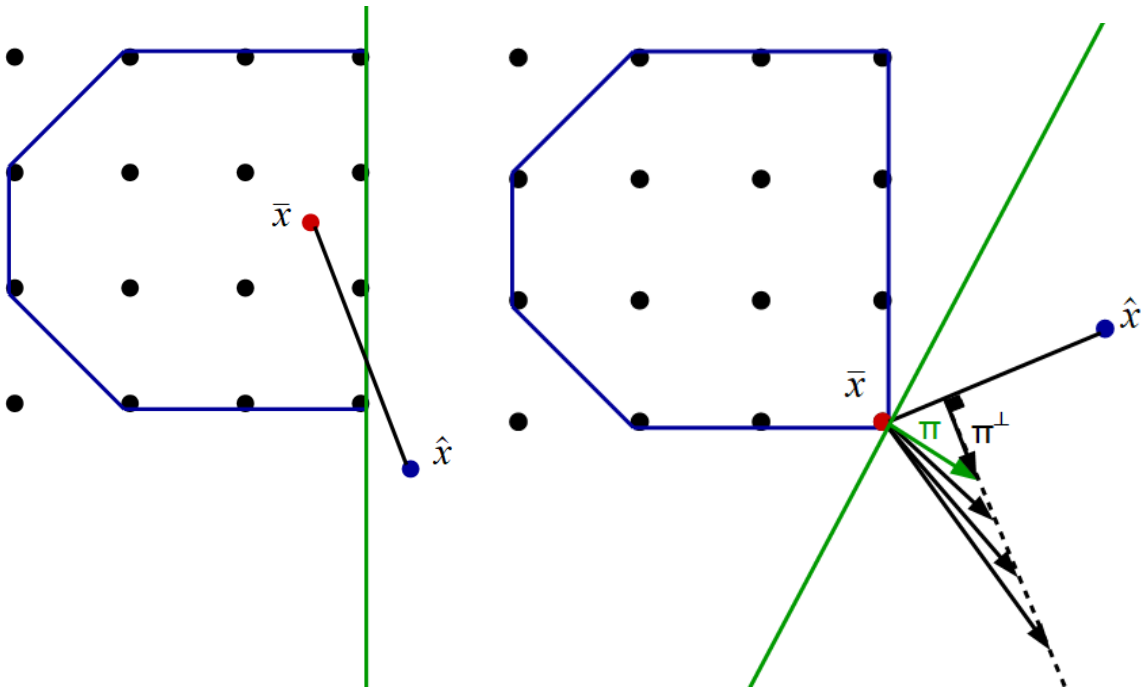
$$\begin{aligned}(D) \quad & \min_{\lambda_i,\ z} |z| \\ & \text{tel que} \\ & \sum_{i\in I} \lambda_i x_i = z\bar{x} + (1-z)\hat{x} \\ & \sum_{i\in I} \lambda_i = 1 \\ & z \in \mathbb{R},\ \lambda_i \in \mathbb{R}^+, \quad \forall i \in I\end{aligned}$$

Ce problème se traduit en langage courant par : trouver le point de Q_2 le plus proche de $\hat{x}$ sur la droite contenant $\bar{x}$ et $\hat{x}$. Le problème (S) est borné lorsque son dual (D) est faisable ce qui arrive lorsqu'il existe un point appartenant à la fois à Q_2 et à la droite précédente. Cette condition peut facilement être remplie en choisissant $\bar{x}$ égal à un point connu de Q_2. Contrairement à la normalisation précédente, ce point n'a pas besoin d'être dans l'intérieur de Q_2. Lorsque $\bar{x}$ appartient à Q_2, la coupe générée est une facette de Q_2 contenant le point d'intersection entre la frontière de Q_2 et le segment reliant $\hat{x}$ et $\bar{x}$. Ceci est illustré en Figure

5.3a. Notons que lorsque $\bar{x}$ se trouve sur la frontière de Q_2, la valeur optimale du problème (S) est bornée mais l'ensemble des solutions optimales peut être non borné. En effet, considérons, l'exemple suivant illustré en Figure 5.3b.

Exemple : Supposons que $\bar{x}$ soit situé sur un sommet de Q_2 et que $\hat{x}$ soit situé de telle manière que la droite $(\hat{x}, \bar{x})$ intersecte Q_2 uniquement en $\bar{x}$. Soit $\pi x \leq \pi\bar{x}$ une coupe optimale séparant $\hat{x}$ de Q_2 et passant par $\bar{x}$. Soit $\pi^\perp$ la projection de π sur l'orthogonal de l'espace vectoriel induit par le vecteur $\hat{x} - \bar{x}$. Alors pour tout $\alpha \geq 0$, la coupe $(\pi + \alpha\pi^\perp)x \leq (\pi + \alpha\pi^\perp)\bar{x}$ est aussi une coupe optimale pour le problème de séparation. La violation de ces coupes par $\hat{x}$ est indépendante de la valeur de α mais l'ensemble des coupes optimales est bien non borné.

Malgré cet ensemble non borné de solutions, seules les facettes du polyèdre Q_2 sont des sommets de l'espace des solutions de (S). Donc, si l'algorithme résolvant (S) renvoie toujours un sommet, la coupe générée sera toujours une facette.



(a) Coupe associée à une normalisation directionnelle

(b) Différentes normales de coupe optimales pour la normalisation directionnelle : la norme des normales optimales peut tendre vers l'infini

FIGURE 5.3 – Exemples de résultats obtenus lors de l'utilisation d'une normalisation directionnelle.

Les normalisations ci-dessus ont été présentées à plusieurs reprises dans la littérature et sont applicables dans le cas général. Cependant, certains problèmes peuvent admettre des normalisations particulièrement adaptées à leur structure. On retrouve de telles normalisations par exemple dans les problèmes de séparation issus de la programmation disjonctive (Balas et Perregaard, 2002). Par ailleurs, le problème de flot insécable possède une normalisation qui semble naturelle et que nous présenterons en Section 5.6.1.

5.4 Une nouvelle approche pour le sous-problème de Fenchel

Dans cette section, nous présentons une nouvelle approche de résolution pour le problème de séparation de la décomposition de Fenchel lorsque la normalisation directionnelle, présentée en Section 5.3, est utilisée. Le programme linéaire de séparation (S) présente des problèmes d'instabilité numérique avec la normalisation directionnelle. En effet, celui-ci cherche une face du polyèdre Q_2 intersectant le segment $(\hat{x}, \bar{x})$. Si par exemple, ce segment intersecte une facette de Q_2 en formant un angle proche de 0 avec cette facette alors une faible erreur sur les paramètres du segment ou de la facette peut induire une grande erreur sur la position du point d'intersection. Pour cette raison, le problème (S) est parfois trop instable numériquement pour être résolu directement. L'intérêt de cette nouvelle approche est de générer la coupe issue de la normalisation directionnelle en utilisant dans le problème (S) une autre normalisation que nous appellerons normalisation alternative. Le problème de séparation associé à cette normalisation alternative est nommé séparation alternative. Si la normalisation alternative utilisée ne présente pas elle même des problèmes d'instabilité numérique, les coupes et les sommets renvoyés possèdent une bonne précision.

La méthode proposée est décrite dans l'Algorithme 3 et illustrée en Figure 5.4. L'idée derrière cette méthode est que la coupe associée à une normalisation directionnelle vers $\bar{x}$ est la même pour $\hat{x}$ et pour tout autre point du segment $(\hat{x}, \bar{x})$ n'appartenant pas à Q_2. En projetant $\hat{x}$ sur des coupes intermédiaires, l'algorithme décale progressivement un point x' le long du segment $(\hat{x}, \bar{x})$ en direction de $\bar{x}$. Une fois que x' atteint la frontière de Q_2 la procédure s'arrête. Notons que cet algorithme calcul des points d'intersections entre le segment $(\hat{x}, \bar{x})$ et des coupes renvoyées par le problème de normalisation alternatif. Si le segment est presque parallèle à l'une de ces coupes, le calcul de ce point d'intersection peut être instable numériquement. Cependant, ce point n'est qu'un intermédiaire de calcul qui n'est pas renvoyé par la méthode. Les résultats de la méthode sont une coupe et des sommets qui sont eux calculés avec une normalisation alternative qui est supposée ne pas présenter d'instabilité numérique. Les résultats de la méthode possède donc un bonne précision.

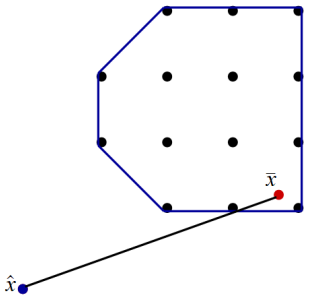
Algorithme 3 Nouvelle approche pour la résolution du sous-problème de Fenchel

Entrées : Q_2 un polyèdre, $\hat{x}$ un point à séparer de Q_2, $\bar{x}$ un point appartenant à Q_2
Sorties : une coupe C séparant $\hat{x}$ de Q_2 et contenant le point d'intersection entre la frontière de Q_2 et le segment $(\hat{x}, \bar{x})$, une liste LS de sommets de Q_2 vérifiant la coupe C à l'égalité

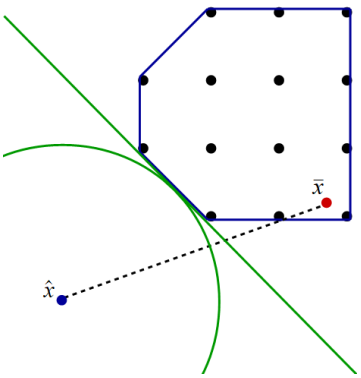
1: Poser x' égal à $\hat{x}$
2: **Tant que** $x' \notin Q_2$ **faire**
3: C, LS = Séparation_alternative(Q_2, x')
4: Poser x' égal à l'intersection entre le segment $(\hat{x}, \bar{x})$ et la coupe C
5: **Renvoyer** C, LS

Dans les sections suivantes, nous allons étudier la question : cette nouvelle méthode converge-t-elle en temps fini ? Nous émettons l'hypothèse que cette méthode converge quelle que soit la normalisation alternative. Cependant, nous ne démontrons cette convergence que dans deux cas particuliers : lorsque la normalisation alternative génère toujours des facettes de Q_2 et lorsque la normalisation alternative est $\|\pi\| \leq 1$ pour une norme quelconque. Ces cas

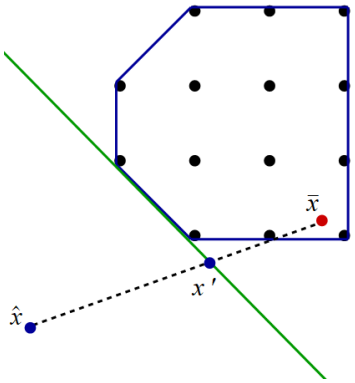
particuliers couvrent l'ensemble des normalisations citées en Section 5.3 mais pas l'ensemble des normalisations possibles.



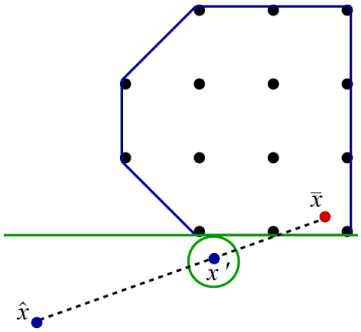
(a) L'objectif est de résoudre le problème de séparation du point bleu avec la normalisation directionnelle en utilisant le point rouge comme point intérieur



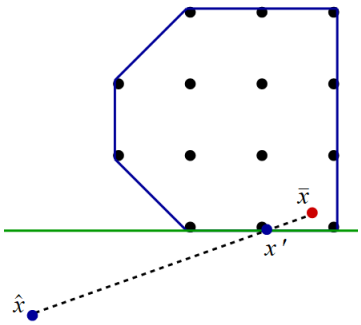
(b) Le problème de séparation est résolu avec la normalisation alternative (ici la norme $\|.\|_2$)



(c) Le point à séparer est projeté sur la coupe générée



(d) Le problème de séparation est de nouveau résolu avec la normalisation alternative (ici la norme $\|.\|_2$)



(e) Une fois que le point à séparé est projeté sur la bordure du polyèdre, la procédure s'arrête

FIGURE 5.4 – Exemple de résolution du sous-problème de Fenchel pour la normalisation directionnelle en utilisant une normalisation alternative

5.4.1 Normalisation alternative générant des facettes

Dans cette section, nous nous intéressons à la terminaison de la méthode proposée pour résoudre le sous-problème de Fenchel lorsque la normalisation alternative utilisée génère des facettes de Q_2.

Théorème 5.2
Si la normalisation alternative utilisée garantit la génération de facettes alors la méthode présentée génère une coupe associée à une normalisation directionnelle en temps fini.

Démonstration. Nous allons montrer que la méthode se termine au plus tard une fois que la séparation alternative a généré toutes les facettes de Q_2. Pour cela, nous montrons que chaque facette de Q_2 est générée au plus une fois. Au fur et à mesure de l'algorithme, le point x' séparé lors des séparations alternatives avance le long du segment $(\hat{x}, \bar{x})$ en direction de $\bar{x}$. Une fois qu'une facette de Q_2 est générée, le point x' est projeté sur cette facette le long du segment $(\hat{x}, \bar{x})$. Les futurs points x' vérifieront donc tous l'inégalité associé à cette facette qui ne pourra donc plus être générée par le problème de séparation alternatif. Puisqu'un polyèdre possède un nombre fini de facettes, une facette intersectant le segment $(\hat{x}, \bar{x})$ est générée en un nombre fini d'étapes. Une fois que cela se produit, la méthode se termine après un unique appel au problème de séparation alternatif. En effet, la procédure place le point x' sur le point d'intersection entre cette facette et le segment $(\hat{x}, \bar{x})$. À l'itération suivante, le problème de séparation alternatif indique que le point x' appartient à Q_2 et la méthode s'arrête. □

5.4.2 Normalisation alternative à l'aide d'une norme quelconque

Nous présentons maintenant une démonstration de convergence lorsque la normalisation alternative est $\|\pi\| \leq 1$ pour une norme quelconque. Cette preuve utilise les Lemmes 5.1 et 5.2 présentés par Boyd (1995). Auparavant, nous rappelons les propriétés et notations suivantes. Dans le cas d'une normalisation $\|\pi\| \leq 1$, le problème dual du problème de séparation est : trouver le point de Q_2 minimisant la distance $\|x - \hat{x}\|^*$ où $\|.\|^*$ est la norme duale de $\|.\|$, $\|\lambda\|^* = \max_{\|x\| \leq 1} \lambda x$. Le point solution du problème dual est noté x^* et vérifie toujours la coupe générée par le problème de séparation à l'égalité. Nous présentons maintenant les lemmes utilisés dans la preuve.

Lemme 5.1
Soit $\lambda^* x \leq \lambda^* x^*$ la coupe générée lors de la séparation de $\hat{x}$ et de Q_2 avec la normalisation $\|\pi\| \leq 1$ où x^* est la solution optimale du problème dual du problème de séparation. Alors $-\lambda^*$ est un sous-gradient de $x \mapsto \|x - \hat{x}\|^*$ en x^*.

Lemme 5.2
Pour chaque norme, il existe un angle $\theta > 0$ tel que en tout point x et pour tout sous-gradient λ de cette norme en x :

$$\angle(\lambda, x) \leq \frac{\pi}{2} - \theta$$

où $\angle(\lambda, x)$ est l'angle entre les vecteurs λ et x.

Théorème 5.3
Si la normalisation alternative utilisée est $\|\pi\| \leq 1$ pour une norme quelconque alors la méthode présentée génère une coupe associée à une normalisation directionnelle en temps fini.

Démonstration. Schéma de la preuve : dans un premier temps nous allons montrer qu'une fois qu'une face intersectant le segment $(\hat{x}, \bar{x})$ a été générée la méthode se termine après un unique appel au problème de séparation alternatif. Dans un second temps, nous montrerons que si une autre face de Q_2 est générée, le point x' avance le long du segment $(\hat{x}, \bar{x})$ en direction de

$\bar{x}$ d'une distance ϵ strictement positive indépendante de l'itération. Ce deuxième cas ne peut donc pas arriver plus de $\frac{\|\bar{x}-\hat{x}\|}{\epsilon}$ fois donc la procédure se termine en un nombre fini d'étapes.

1) Supposons qu'à une itération, une face intersectant le segment $(\hat{x}, \bar{x})$ est générée. Après avoir généré la face, le procédure place le point x' sur le point d'intersection. A l'itération suivante, le problème de séparation alternatif indique alors que le point x' appartient à Q_2 et la méthode s'arrête.

Pour la suite de cette preuve, nous noterons $x^{(i)}$ le point x' généré à l'itération i de l'algorithme.

2) Nous allons montrer que si une face F de Q_2 n'intersectant pas le segment $(\hat{x}, \bar{x})$ est générée, le point x' avance le long du segment $(\hat{x}, \bar{x})$ d'une distance strictement positive, c'est à dire $\|x^{(i+1)} - x^{(i)}\|_2 \geq \epsilon$ pour un certain $\epsilon > 0$ indépendant de la face F.

2.1) Supposons qu'à l'itération i, le problème de séparation alternatif sur $x^{(i)}$ renvoie une coupe $\lambda^* x \leq \lambda^* x^*$ où x^* est la solution optimale du problème dual du problème de séparation. D'après le Lemme 5.1, puisque la normalisation alternative est $\|\pi\| \leq 1$, le vecteur $-\lambda^*$ est un sous-gradient de $\|x - x^{(i)}\|^*$ en x^*. Donc d'après le Lemme 5.2, on a $\angle(-\lambda^*, x^* - x^{(i)}) \leq \frac{\pi}{2} - \theta$ pour un certain $\theta > 0$ dépendant de la norme étudiée mais pas de la face F. Après avoir généré la coupe $\lambda^* x \leq \lambda^* x^*$, la procédure projette le point $x^{(i)}$ sur l'hyperplan $\lambda^* x = \lambda^* x^*$ le long du segment $(\hat{x}, \bar{x})$. Le résultat de cette projection est le point $x^{(i+1)}$. Le point $x^{(i+1)}$ est sur l'hyperplan $\lambda^* x = \lambda^* x^*$ dont la normale λ^* vérifie $\angle(\lambda^*, x^{(i)} - x^*) \leq \frac{\pi}{2} - \theta$ donc l'angle $\angle(x^{(i+1)} - x^*, x^{(i)} - x^*)$ est supérieur à θ.

2.2) Soit d_{min} la distance en norme $\|.\|_2$ entre le segment $(\hat{x}, \bar{x})$ et l'union des faces de Q_2 n'intersectant pas le segment $(\hat{x}, \bar{x})$. Puisque la face F n'intersecte pas le segment $(\hat{x}, \bar{x})$, la distance entre le segment $(\hat{x}, \bar{x})$ et la face F est supérieure à d_{min}. Or $x^{(i)}$ appartient au segment $(\hat{x}, \bar{x})$ et x^* à la face F donc la distance entre $x^{(i)}$ et x^* est supérieure à d_{min}.

2.3) Nous allons montrer que $\|x^{(i)} - x^{(i+1)}\|_2 \geq d_{min} \sin(\theta)$. Soit D la droite passant par x^* et $x^{(i+1)}$. Le point de la droite D le plus proche de $x^{(i)}$ est à une distance de $x^{(i)}$ de $\|x^{(i)} - x^*\|_2 \sin(\angle(x^{(i)} - x^*, x^{(i+1)} - x^*))$. Or nous avons vu plus haut dans le paragraphe 2.1) que l'angle $\angle(x^{(i)} - x^*, x^{(i+1)} - x^*)$ est supérieur à θ et que le point $x^{(i+1)}$ est plus loin de $x^{(i)}$ que le point le plus proche de la droite D. Donc la distance entre $x^{(i)}$ et $x^{(i+1)}$ vérifie : $\|x^{(i)} - x^{(i+1)}\|_2 \geq \|x^{(i)} - x^*\|_2 \sin(\angle(x^{(i)} - x^*, x^{(i+1)} - x^*)) \geq \|x^{(i)} - x^*\|_2 \sin(\theta)$. De plus, nous avons vu dans le paragraphe 2.2) que la distance entre $x^{(i)}$ et x^* est supérieure à d_{min}. Donc finalement on a bien : $\|x^{(i)} - x^{(i+1)}\|_2 \geq d_{min} \sin(\theta)$.

La distance $d_{min} \sin(\theta)$ est bien strictement positive et indépendante de l'itération de l'algorithme ce qui complète la preuve.

□

La nouvelle méthode de résolution pour le problème de Fenchel que nous venons de présenter fonctionne lorsque la plupart des normalisations connues de la littérature sont utilisées

en normalisation alternative. Cependant, nous faisons l'hypothèse que les preuves de convergences ci-dessus peuvent être étendues à d'autres normalisations alternatives. Par ailleurs, il serait intéressant d'étendre cette méthode de résolution pour résoudre le sous-problème de Fenchel pour d'autres normalisations que la normalisation directionnelle.

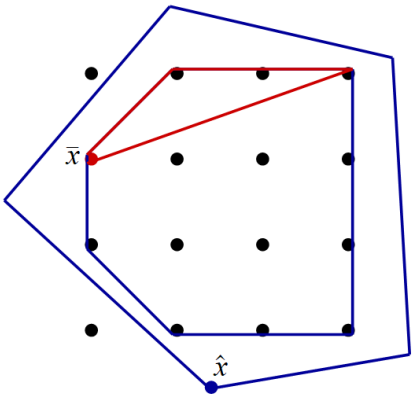
5.5 Couplage des décompositions de Fenchel et de Dantzig-Wolfe

Dans cette section, nous présentons une méthode de décomposition combinant une décomposition de Dantzig-Wolfe et une décomposition de Fenchel. L'idée motivant cette approche est la suivante. Lors de la génération d'une coupe de Fenchel, les variables primales du sous-problème de séparation (S) sont les coefficients de la contrainte générée et les contraintes actives correspondent aux sommets du polyèdre séparé Q_2 vérifiant cette coupe à l'égalité. Ainsi, le sous-problème de Fenchel génère à la fois des coupes valides pour le polyèdre Q_2 et des sommets de ce polyèdre. L'idée est d'utiliser les coupes générées pour améliorer une relaxation linéaire de Q_2 tandis que les sommets sont utilisés dans une formulation de Dantzig-Wolfe afin d'améliorer une approximation interne de Q_2. L'un des points clés de la méthode est de séparer la solution $\hat{x}$ du problème maître de Fenchel à l'aide d'une normalisation directionnelle. Cette normalisation nécessite la connaissance d'un point dans le polyèdre Q_2 et nous décidons d'utiliser la solution $\bar{x}$ du problème maître de Dantzig-Wolfe.

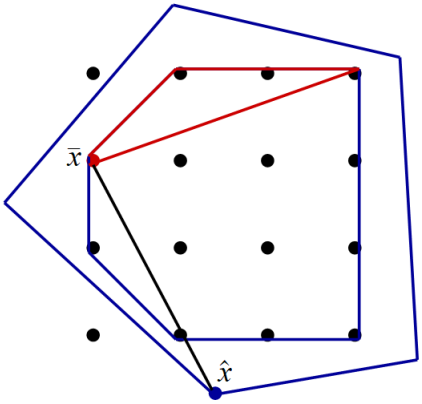
Intuitivement, les formulations de Fenchel et de Dantzig-Wolfe sont en désaccord sur l'emplacement de la frontière de Q_2 sur le segment $(\hat{x}, \bar{x})$. Le problème de séparation trouve la position exacte de la frontière et donne aux deux formulations des informations leur permettant de rendre leur approximation de Q_2 exacte en ce point. Une autre manière de voir la méthode est de dire qu'elle améliore progressivement le point $\bar{x}$. A chaque itération, l'algorithme teste une direction de potentielle amélioration $\hat{x} - \bar{x}$. La méthode trouve alors soit de nouveaux sommets de Q_2 permettant d'améliorer $\bar{x}$ soit une facette de Q_2 passant par $\bar{x}$ prouvant qu'il n'est pas possible d'améliorer $\bar{x}$ dans cette direction. Cette interaction entre un point extérieur et un point intérieur de Q_2 n'est pas sans rappeler la séparation *in-out* proposée pour la décomposition de Benders (Ben-Ameur et Neto, 2007).

La méthode de décomposition comprend donc deux formulations du problème fonctionnant en tandem. La première formulation est la formulation de Fenchel (F) dans laquelle sont rajoutées les coupes de Fenchel générées. La deuxième formulation est la formulation de Dantzig-Wolfe (DW) dans laquelle sont rajoutés les sommets générés dans le sous-problème de séparation. Ainsi les étapes de l'algorithme, illustrées dans la Figure 5.5, sont les suivantes :

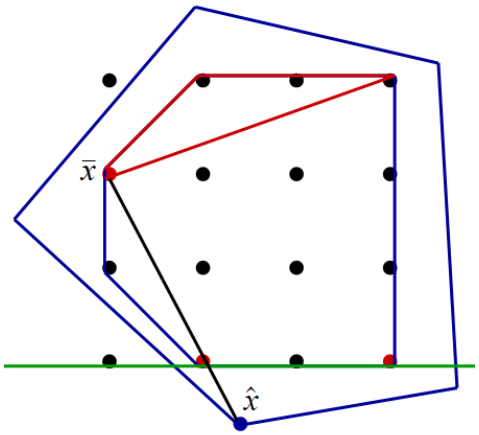
1. Résoudre de le problème (F) avec un sous-ensemble des coupes de Fenchel : une solution $\hat{x}$ est obtenue dont la valeur est une borne supérieure du problème.
2. Résoudre de la formulation de Dantzig-Wolfe (DW) avec un sous-ensemble des sommets de Q_2 : une solution $\bar{x}$ est obtenue dont la valeur est une borne inférieure du problème.
3. Si les deux bornes sont égales : fin de l'algorithme



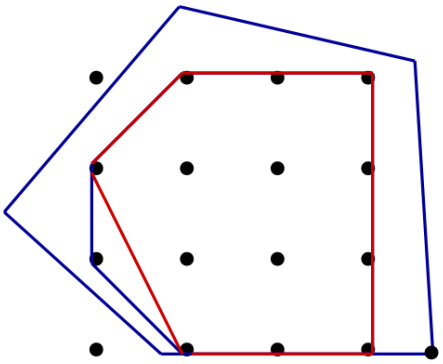
(a) Les solutions optimales de la formulation de Dantzig-Wolfe (point rouge) et de la formulation de Fenchel (point bleu) sont calculées



(b) Le problème de séparation de la solution optimale de la formulation de Fenchel est résolu en utilisant la normalisation directionnelle avec la solution de la formulation de Dantzig-Wolfe comme point intérieur



(c) Le problème de séparation renvoie une coupe et des sommets du polyèdre à séparer



(d) La coupe est ajoutée à la formulation de Fenchel et les sommets à la formulation de Dantzig-Wolfe

FIGURE 5.5 – Différentes étapes d'une itération de la décomposition Dantzig-Wolfe-Fenchel

4. Séparer le point $\hat{x}$ de Q_2 à l'aide du problème de séparation (S) en utilisant une normalisation directionnelle avec $\bar{x}$ comme point intérieur : une coupe est obtenue ainsi que des sommets de Q_2.
5. Ajouter la coupe au problème (F) et les sommets à la formulation de Dantzig-Wolfe (DW).
6. Reprendre à l'étape 1

Par rapport à une décomposition de Fenchel classique, cette méthode concentre sa géné-

ration de coupes autour du point $\bar{x}$. Elle affine ainsi la connaissance du polyèdre Q_2 autour de ce point. D'autre part, comparée à une méthode de Dantzig-Wolfe classique, cette méthode consacre un temps plus important à la résolution de son sous-problème ce qui lui permet de générer un plus grand nombre de sommets à rajouter dans le problème maître.

5.6 Application au problème de flot insécable

5.6.1 Normalisation naturelle pour les flots insécables

Soit $\hat{f} = (\hat{f}^k)_{k \in K} \in [0,1]^{|K|}$ une distribution de flot pour chaque commodité induisant un dépassement de capacité $\hat{o}$ sur un arc. Une question venant naturellement est : quelle est la décomposition de cette distribution en patron de commodités induisant un dépassement de capacité minimum ? Cette question peut être résolue en utilisant le programme linéaire de décomposition suivant :

$$
\begin{aligned}
(D) \quad & \min_{\lambda^g, z} z \\
& \text{tel que} \\
& \sum_{i \in I} \lambda^g f^g = \hat{f} \\
& \sum_{i \in I} \lambda^g o^g = \hat{o} + z \\
& \sum_{i \in I} \lambda^g = 1 \\
& \lambda^g \in \mathbb{R}^+, \ z \in \mathbb{R}^+ \quad \forall i \in I
\end{aligned}
$$

où λ^g est le coefficient dans la décomposition associé à un patron de commodités f^g induisant un dépassement de capacité o^g.

Or le dual de ce programme de décomposition est le programme de séparation suivant :

$$
\begin{aligned}
(S) \quad & \max_{\pi, \pi_o \pi_0} \pi \hat{f} + \pi_o \hat{o} - \pi_0 \\
& \text{tel que} \\
& \pi f^g + \pi_o o^g \leq \pi_0, \quad \forall (f^g, o^g) \in Q_2 \\
& \pi_o = -1 \\
& \pi, \pi_0, \pi_o \in \mathbb{R}
\end{aligned}
$$

Ce programme correspond exactement au problème (S) de séparation du point $\hat{x} = (\hat{f}_1, ..., \hat{f_K}, \hat{o})$ avec une contrainte de normalisation imposant que le coefficient π_o associé à la variable de dépassement de capacité vérifie $\pi_o = -1$. Cette contrainte de normalisation

est ce que nous appellerons la normalisation naturelle pour le problème de flot insécable. Cette normalisation est très proche d'un cas particulier de normalisation directionnelle pour la direction $\bar{x} - \hat{x} = (0, ..., 0, 1)$. Tout comme la normalisation directionnelle, la normalisation naturelle garantie la génération de facette car elle est imposée à l'aide d'une unique contrainte linéaire.

5.6.2 Résolution de l'oracle de sac à dos

Toutes les méthodes de décomposition présentées dans ce chapitre font l'hypothèse qu'il existe un oracle (un algorithme efficace) capable de résoudre le problème d'optimisation d'une fonction linéaire sur le polyèdre Q_2. Dans cette section, nous détaillons quel est exactement le problème résolu par l'oracle dans le cadre des flots insécables ainsi que l'algorithme utilisé pour résoudre ce problème.

Dans la version du problème de flot insécable que nous étudions, les contraintes de capacité n'imposent pas que le flot des commodités respecte les capacités c_e des arcs. Il est uniquement imposé que le dépassement de capacité soit stocké dans une variable o_e. Ainsi, le polyèdre des variables vérifiant la contrainte de capacité associée à l'arc e s'écrit :

$$\{(f_e^k \in \{0,1\})_{e\in E, k\in k} o_e \in \mathbb{R}^+ | \sum_{k\in K} f_e^k d^k \leq c_e + o_e\}.$$

L'optimisation d'une fonction linéaire dont les coefficients sont $(\pi^k)_{k\in K}$ et $-\pi_o$ sur ce polyèdre s'écrit donc :

$$\begin{aligned}
(O_e) \quad & \max_{f_e^k, o_e} \sum_{k\in K} \pi^k f_e^k - \pi_o o_e \\
& \text{tel que} \\
& \sum_{k\in K} f_e^k d^k \leq c_e + o_e \\
& f_e^k \in \{0,1\},\ o_e \in \mathbb{R}^+
\end{aligned}$$

Ce problème n'est pas un problème de sac à dos classique. Cependant, il peut être résolu en résolvant deux problèmes de sac à dos classiques à l'aide d'une disjonction de cas. Cette méthode a été présentée par Büther et Briskorn (2012) mais nous en faisons le rappel ici. Considérons la disjonction de cas suivante : soit le flot des commodités respecte la capacité de l'arc e, soit le flot des commodités dépasse la capacité de l'arc e. Trouver la meilleure solution

dans le premier cas revient à résoudre le problème suivant :

$$\begin{aligned}
&\max_{f_e^k} \sum_{k\in K} \pi^k f_e^k \\
&\text{tel que} \\
&\sum_{k\in K} f_e^k d^k \leq c_e \\
&f_e^k \in \{0,1\}
\end{aligned}$$

En effet, on suppose que le coefficient π_o est positif car dans le cas contraire le problème O_e serait non borné. Puisque le flot des commodités respecte la capacité de l'arc, la variable o_e prend toujours la valeur zéro et peut être retirée du problème. On remarque que dans ce premier cas de la disjonction, le problème à résoudre est un problème de sac à dos classique. Dans le deuxième cas de la disjonction, trouver la meilleure solution revient à résoudre le problème suivant :

$$\begin{aligned}
&\max_{f_e^k, o_e} \sum_{k\in K} \pi^k f_e^k - \pi_o o_e \\
&\text{tel que} \\
&\sum_{k\in K} f_e^k d^k \leq c_e + o_e \\
&\sum_{k\in K} f_e^k d^k \geq c_e \\
&f_e^k \in \{0,1\},\ o_e \in \mathbb{R}^+
\end{aligned}$$

Puisqu'on suppose que π_o est positif et que le flot des commodités ne respecte pas la capacité des arcs, la variable o_e est toujours égale à $\sum_{k\in K} f_e^k d^k - c_e$. En effectuant le remplacement dans la fonction objectif, en remplaçant les variables f_e^k par leur complémentaire $\bar{f}_e^k = 1 - f_e^k$ et en multipliant la contrainte $\sum_{k\in K} f_e^k d^k \geq c_e$ par -1 on obtient la reformulation suivante :

$$\begin{aligned}
&\max_{\bar{f}_e^k} \sum_{k\in K} (\pi_o d^k - \pi^k) \bar{f}_e^k + C \\
&\text{tel que} \\
&\sum_{k\in K} \bar{f}_e^k d^k \leq \sum_{k\in K} d^k - c_e \\
&\bar{f}_e^k \in \{0,1\}
\end{aligned}$$

où la constante C est égale à $\sum_{k\in K}(\pi^k - \pi_o d^k) - \pi_o c_e$. Cette reformulation montre que le problème à résoudre dans le second cas de la disjonction est aussi un problème de sac à dos

classique.

Remarque : dans l'étude experimentale de la Section 5.7, afin de résoudre ces deux problèmes de sac à dos, nous employons l'algorithme MINKNAP proposé par Pisinger (1997). Cependant, cet algorithme ne prend en entrée que des valeurs entières pour les coefficients du problème de sac à dos à résoudre. Or, les coefficients π^k et π_o sont les valeurs de variables duales associées à des problèmes maîtres et ne sont donc pas forcément entiers. Afin de les rendre entiers, ces coefficients sont multipliés uniformément par une large constante et tronqués. Ceci permet de résoudre les problèmes de sac à dos de manière approximative mais avec une bonne précision.

5.7 Étude expérimentale

Dans cette section, nous présentons une comparaison expérimentale de différentes méthodes de décomposition. Les jeux de données et le code utilisés dans cette section sont accessibles à `https://github.com/SuReLI/thesis_decomposition_code`. Le code a été écrit en Python 3 et les expériences ont été faites sur un serveur contenant 16 CPU Intel Core i9-9900K 3.60 GHz, 60 Gbit de RAM et Ubuntu 20.10.

5.7.1 Jeux d'instances

Le graphe et la liste des commodités des instances sont créés avec la méthode pour les instances du problème de flot insécable statique présentée en Section 3.4.

Notons que les demandes créées de cette manière peuvent toujours être routées en respectant les capacités des arcs sans dépassement de capacité. De ce fait, la borne inférieure donnée par la relaxation linéaire est optimale. Afin de créer un saut d'optimalité dans les instances, nous modifions légèrement les capacités de certains arcs de la manière suivante un nombre de fois égale à 100 fois le nombre de nœuds :

— Sélectionner aléatoirement une origine.

— Sélectionner aléatoirement deux arcs sortant de cette origine.

— Ajouter 1 à la capacité d'un des arcs et retirer 1 à la capacité de l'autre arc.

À cause de la procédure utilisée pour créer les commodités, les arcs sortants des origines sont saturés dans les solutions sans dépassement de capacité alors que les autres arcs sont souvent non saturés. De ce fait, dans la plupart des cas, transférer de la capacité d'un arc sortant d'une origine à un autre ne change pas la valeur de la relaxation linéaire. En effet, une partie du flot de l'une des commodités est transférée de l'un des arcs vers l'autre. En revanche, ce transfert de capacité peut avoir un impact sur la valeur de la meilleure solution insécable. En effet, il n'existe plus forcément de combinaison de commodités dont la somme

des demandes est exactement égale à la capacité de chaque arc. Dans ce cas, la meilleure solution insécable possède un dépassement de capacité non nul.

Les jeux de données Trois jeux de données différents sont utilisés au cours des expériences dans lesquelles dix instances sont générées pour chaque valeur du paramètre variable. Dans tous les jeux de données, lors du choix de la demande des commodités, la formule $d = \min(c_p, U(\hat{d}_{max}))$, présentée en Section 3.4, est utilisée.

— Jeu de données *faible demande maximale* : ce jeu de données considère les graphes aléatoires fortement connexes de 50 nœuds à 145 nœuds. La demande maximale des commodités est fixée à $\hat{d}_{max} = 100$ et la capacité des arcs à 1000. Ce choix de demande maximale implique qu'un bon nombre de commodités peut passer par chaque arc. Cependant, dans nos tests, la solution optimale de ces instances ne contient souvent pas de dépassement de capacité. Ces instances ne contiennent donc pas de saut d'optimalité. Notre hypothèse est que le nombre important de commodités leur permet de se réorganiser pour combler exactement la capacité de chaque arc.

— Jeu de données *forte demande maximale* : ce jeu de données considère les graphes aléatoires fortement connexes de 145 nœuds à 1000 nœuds. La demande maximale des commodités est fixée à $\hat{d}_{max} = 1000$ et la capacité des arcs à 1000. A cause de ce choix de demande maximale, ces instances ne contiennent qu'un faible nombre de commodités. Cependant, elles possèdent généralement un saut d'optimalité ce qui nous permet d'étudier l'évolution des bornes inférieures données par les algorithmes.

— Jeu de données *taille des capacités* : ce jeu de données considère des graphes aléatoires fortement connexes de 70 nœuds. La demande maximale des commodités $\hat{d}_{max}$ est fixées à 1/10 de la capacité des arcs qui elle varie de 100 à 100 000. Le problème de sac à dos est connu pour posséder des algorithmes de résolutions pseudo-polynomiaux en la capacité du sac à dos tel que l'algorithme MINKNAP que nous utilisons. Dans le cas des flots insécables, cette capacité du sac à dos correspond à la capacité des arcs. Les instances de ce jeu de données ont toutes la même structure (même taille de graphe, même taille des commodités relativement à la capacité des arcs) mais nous faisons varier les capacités des arcs, *i.e.* le nombre de chiffres significatifs des capacités des arcs et des demandes des commodités. Ceci impacte le temps de résolution de l'algorithme MINKNAP.

5.7.2 Les méthodes de décomposition étudiées

Dans la suite, nous comparons expérimentalement les méthodes de décomposition suivantes :

Fenchel : méthode de décomposition de Fenchel, les coupes générées sont rajoutées à la relaxation linéaire tandis que les sommets générés sont rajoutés à une formulation de Dantzig-Wolfe. Le sous-problème de Fenchel est résolu avec la normalisation naturelle présentée en Section 5.6.1. Les deux formulations n'agissent donc pas en tandem.

Fenchel-sans-pré-traitement : similaire à la méthode *Fenchel* à l'exception que les techniques de réduction de la dimensionnalité et de *lifting* présentées en Section 5.2.4 n'est pas

utilisée dans la résolution du sous-problème.

DW-Fenchel : méthode combinant les décompositions de Fenchel et de Dantzig-Wolfe présentée en Section 5.5, les coupes générées sont rajoutées à la relaxation linéaire tandis que les sommets générés sont rajoutés à une formulation de Dantzig-Wolfe. Le sous-problème de Fenchel est résolu avec la normalisation directionnelle avec le point optimale de la formulation de Dantzig-Wolfe comme point intérieur. L'utilisation de cette normalisation couplent les deux formulations.

DW-Fenchel-sans-pré-traitement : similaire à la méthode *DW-Fenchel* à l'exception que les techniques de réduction de la dimensionnalité et de *lifting* présentées en Section 5.2.4 n'est pas utilisée dans la résolution du sous-problème.

DW-Fenchel-itératif : similaire à la méthode *DW-Fenchel* à l'exception que le sous-problème de Fenchel est résolu à l'aide de la méthode itérative présentée en Section 5.4.

DW : méthode de décomposition de Dantzig-Wolfe. Aucune stabilisation des variables duales n'est utilisée. Les bornes inférieures sont calculées à l'aide des variables duales et de la valeur de la solution des sous-problèmes de sac à dos.

DW-momentum : similaire à la méthode *DW* à l'exception que les variables duales sont stabilisées à l'aide de la méthode de lissage de Neame (2000) présentée en Section 5.1.2.

DW-point-intérieur : similaire à la méthode *DW* à l'exception que le problème maître de Dantzig-Wolfe est résolu avec une méthode de points intérieurs afin de retourner une solution non optimale mais centrée. Ceci a pour effet de stabiliser les variables duales.

5.7.3 Paramètres des algorithmes

Paramètres du solveur de points intérieurs Afin de calculer un point intérieur proche de la solution d'un programme linéaire, nous demandons au solveur (Gurobi Optimization, 2020) de résoudre le programme linéaire à l'aide d'une méthode de point intérieur avec une précision de 10^{-3} et sans utiliser de méthode de *Cross over*. La première fois que le sous-problème n'arrive pas à générer de nouvelle variable de coût réduit négatif, le solveur (Gurobi Optimization, 2020) est remis à ses paramètres par défaut afin de garantir un calcul exact des derniers coûts réduits. Avec les paramètres par défaut, la génération de colonnes n'est plus stabilisée.

Précision du solveur de sac à dos La constante multiplicative utilisée pour tronquer les paramètres de l'instance donnée au solveur MINKNAP est définie égale à 10^7. Cela correspond à une précision de 10^{-7} sur les paramètres de l'instance.

Chemins autorisés Afin de ne pas avoir à générer des chemins et pouvoir considérer des instances de plus grande taille nous ne considérons pas tous les chemins possibles pour chaque commodité. L'ensemble des chemins autorisés pour une commodité est composé des quatre plus courts chemins entre son origine et sa destination ainsi que du chemin utilisé pour créer la commodité (voir Section 3.4).

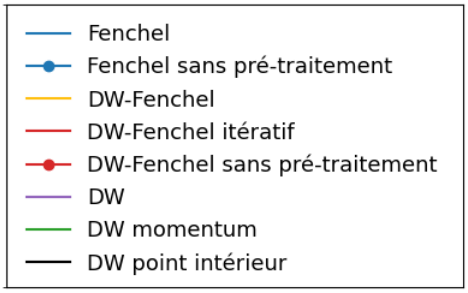


FIGURE 5.6 – Légende des Figures 5.7 à 5.13

Condition de terminaison des algorithmes Les méthodes de décomposition considérées sont arrêtées lorsque l'écart entre leurs bornes est de 10^{-3}.

Stabilisation par momentum Le coefficient de stabilisation α dans la formule de Neame (2000) $\hat{\pi}_t = \alpha\hat{\pi}_{t-1} + (1-\alpha)\pi_t$ est défini égal à 0.8 dans nos tests.

Pré-traitements pour le sous problème de Fenchel : sauf mentionné contraire, tout les algorithmes utilisent les techniques de réduction de la dimensionnalité et de *lifting* présentées en Section 5.2.4 car elles améliorent grandement les performances. En revanche, les autres pré/post-traitement de cette section n'ont pas été implémentés.

5.7.4 Résultats expérimentaux

Dans chaque figure, nous affichons l'évolution des bornes inférieure et supérieure données par les algorithmes en fonction du temps de calcul (en secondes). Notons que toutes les valeurs affichées ne sont pas directement les bornes mais leur écart à la valeur d'une solution optimale de la reformulation de Dantzig-Wolfe (qui est aussi la borne calculée lorsque les autres méthodes convergent). Les courbes tracées représentent les résultats moyens des algorithmes agrégés sur des instances utilisant les mêmes paramètres, tandis que les intervalles de confiance à 95% pour la moyenne sont représentés en semi-transparence autour de la courbe principale. Ces intervalles de confiance sont créés à l'aide de la méthode statistique appelée *Bootstrapping* avec un nombre de rééchantillonnages égal à 1000. Les algorithmes ne renvoient des bornes qu'à la fin de chacune de leurs itérations. Or, ces itérations prennent un temps variable en fonction des instances pour un même algorithme. Il n'est donc pas possible d'agréger directement les courbes des bornes qui doivent être rééchantillonnées. Ce rééchantillonnage est fait toutes les dix secondes en considérant que les bornes évoluent linéairement entre deux itérations. Les tremblements des intervalles de confiance sont dus à ce rééchantillonnage et à celui du *Bootstrapping*. Ils peuvent être interprétés comme l'incertitude sur la borne des intervalles de confiance due à la méthode du *Bootstrapping*.

Impact du pré-traitement. Nous démontrons l'impact du pré-traitement décrit dans le paragraphe "Réduction de la dimensionnalité" de la Section 5.2.4 en comparant la méthode

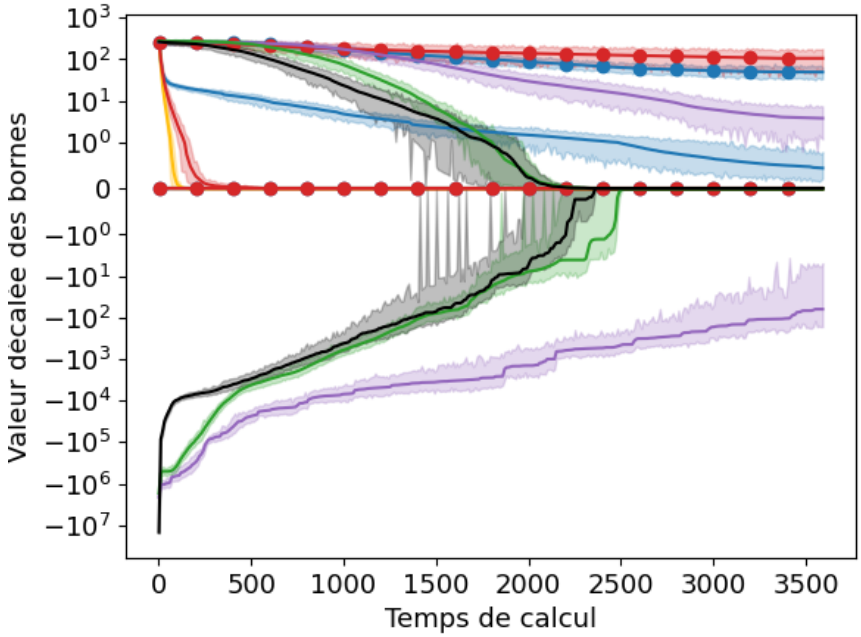


FIGURE 5.7 – Jeu de données *Faible demande maximale*, 70 nœuds

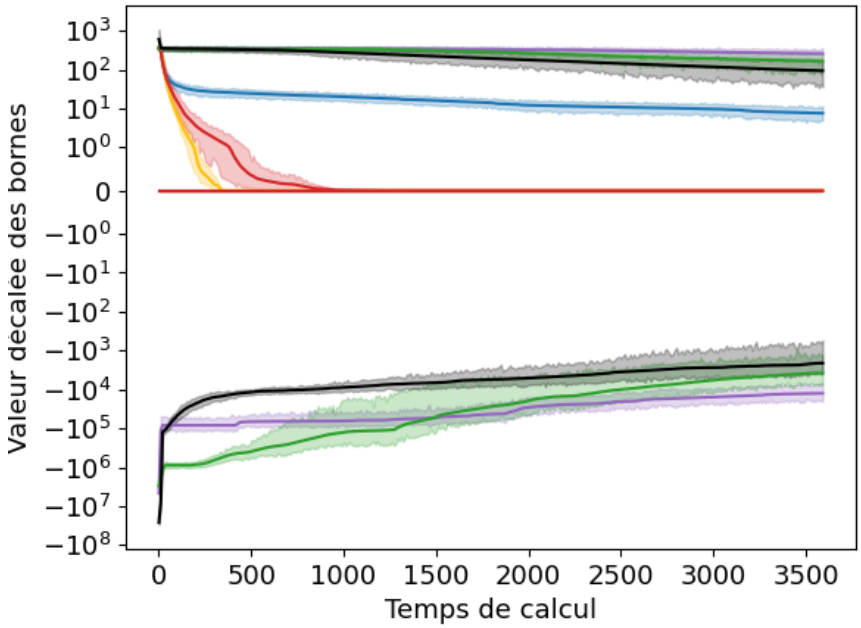


FIGURE 5.8 – Jeu de données *Faible demande maximale*, 90 nœuds

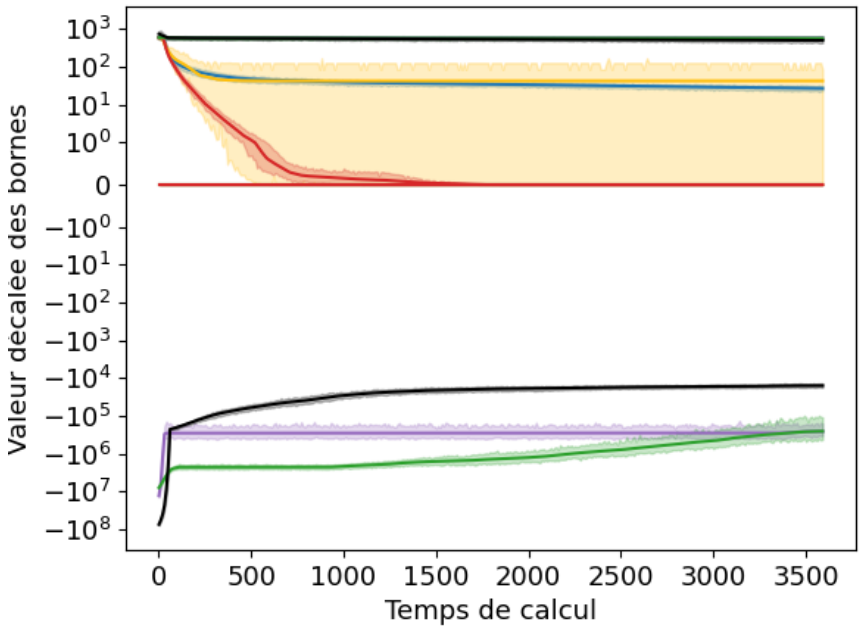


FIGURE 5.9 – Jeu de données *Faible demande maximale*, 145 nœuds

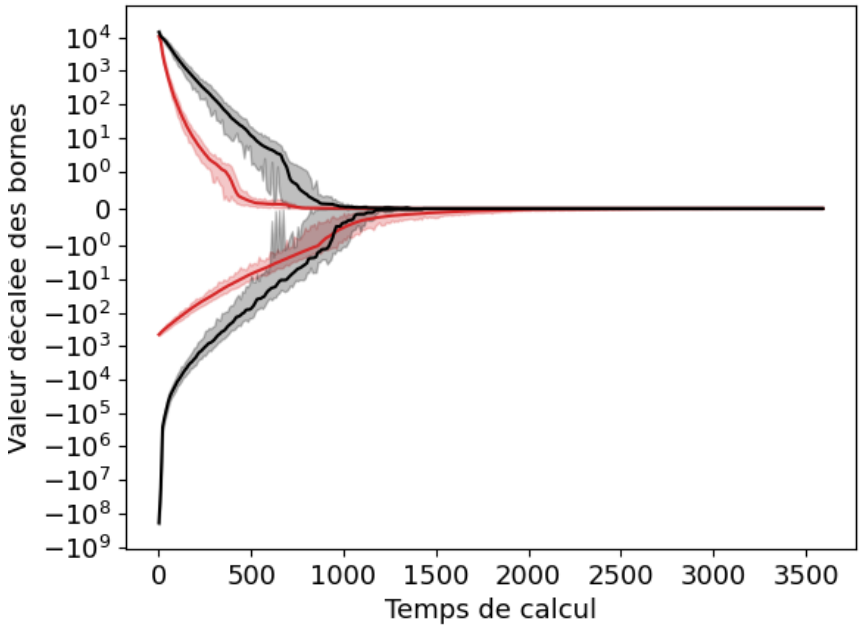


FIGURE 5.10 – Jeu de données *Forte demande maximale*, 250 nœuds

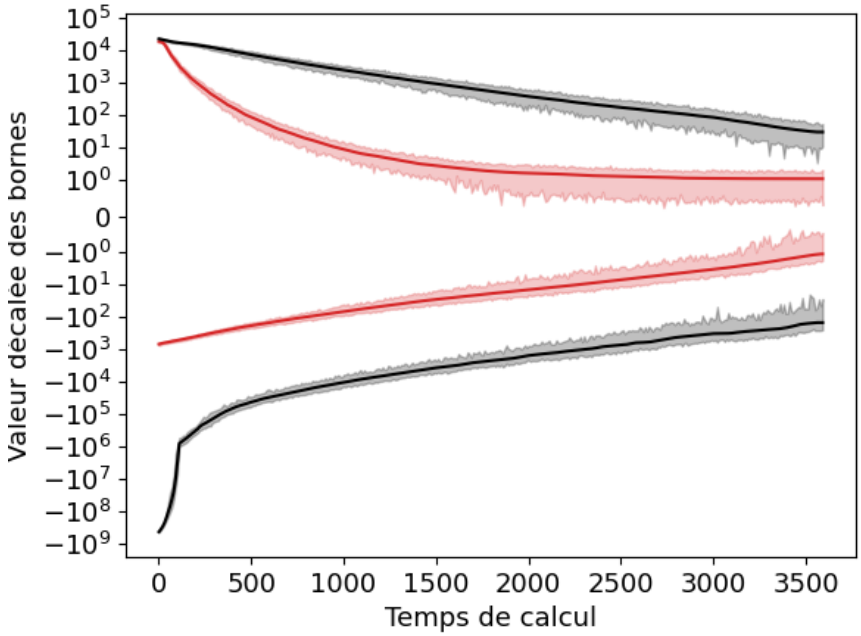


FIGURE 5.11 – Jeu de données *Forte demande maximale*, 400 nœuds

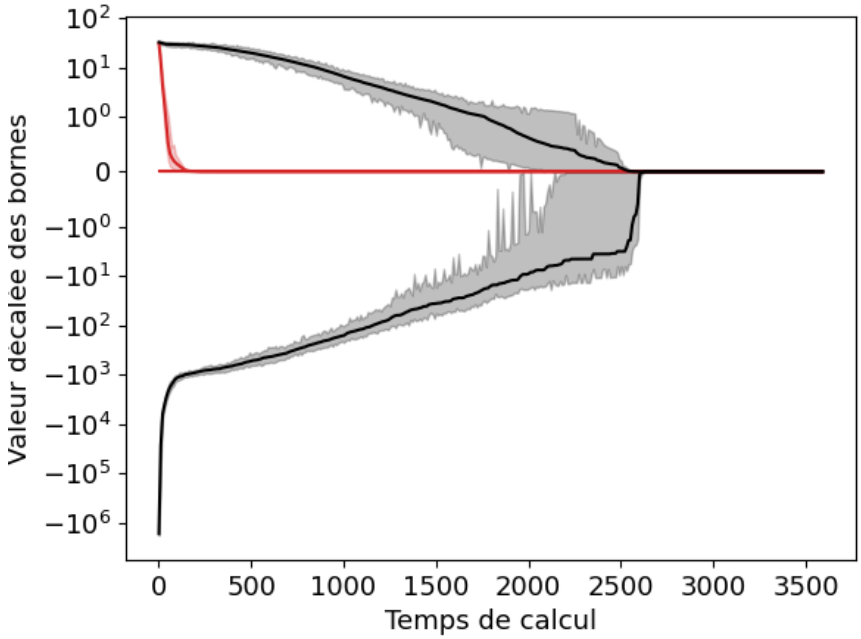


FIGURE 5.12 – Jeu de données *taille des capacités*, capacité 100

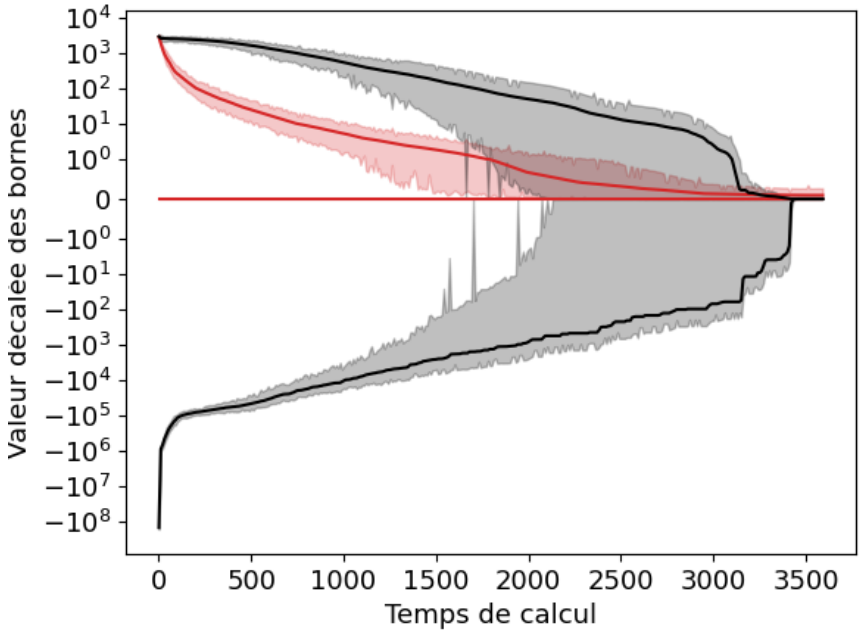


FIGURE 5.13 – Jeu de données *taille des capacités*, capacité 10000

Fenchel à Fenchel-sans-pré-traitement et la méthode DW-Fenchel-itératif à DW-Fenchel-sans-pré-traitement. On constate en Figure 5.7 que même pour de petites instances contenant uniquement 70 nœuds, les méthodes sans pré-traitement ne parviennent pas à converger car elles ont des difficultés à résoudre le sous-problème de Fenchel. Ces méthodes ne sont pas testées sur les grandes instances.

Impact de la stabilisation sur la décomposition de Dantzig-Wolfe. Nous utilisons deux méthodes de stabilisation dans nos tests : la stabilisation par lissage de Neame (2000) ainsi que la stabilisation par point intérieur. On constate qu'en l'absence de stabilisation, la décomposition de Dantzig-Wolfe présente des difficultés de convergence. Ce problème est allégé par les deux méthodes de stabilisation proposées, en particulier avec la stabilisation par point intérieur qui donne régulièrement les meilleurs résultats.

Résolution du sous-problème de Fenchel avec la normalisation alternative. La nouvelle méthode de résolution du sous-problème de Fenchel présentée en Section 5.4 est utilisée dans la méthode DW-Fenchel-itératif. En comparant les résultats de cette méthode avec ceux de DW-Fenchel qui utilise une approche directe pour résoudre le sous-problème, on constate que la nouvelle méthode est plus lente que l'approche directe. En revanche, comme on peut le voir en Figure 5.9, il arrive que l'approche directe échoue à résoudre le sous-problème à cause d'instabilités numériques ce qui empêche la méthode de décomposition de converger.

Impact du couplage des deux problèmes maîtres à l'aide de la normalisation directionnelle. Les méthodes DW-Fenchel et DW-Fenchel-itératif couplent les problèmes maître de Dantzig-Wolfe et de Fenchel à l'aide de l'utilisation d'une normalisation directionnelle dans le sous-

problème de Fenchel. L'impact de ce couplage peut être étudié en comparant ces méthodes à la méthode Fenchel dont l'unique différence est d'utiliser la normalisation naturelle du problème de flot insécable dans son sous-problème. Comme illustré en Figure 5.8, on constate que toutes les méthodes sont similaires au cours des premières itérations. Cependant, la méthode Fenchel ralentit fortement au milieu de sa convergence et les itérations suivantes n'impactent que faiblement les bornes. En revanche, ce n'est pas du tout le cas des méthodes utilisant la normalisation directionnelle qui présente de bien meilleurs résultats. Notre interprétation de ce phénomène est la suivante. Il existe de nombreuses solutions optimales équivalentes de la relaxation linéaire du problème de flot insécable. Lorsqu'une coupe est générée avec la normalisation naturelle, elle ne coupe qu'un sous-ensemble de ces solutions et la nouvelle solution du problème maître de Fenchel se situe à un endroit complètement différent de l'espace des solutions. La méthode ne parvient alors pas à couper toutes les solutions à cause de ce grand nombre de symétries. En revanche, les coupes générées à l'aide de la normalisation directionnelle se concentrent autour de la solution optimale du problème maître de Dantzig-Wolfe dont elles essayent de démontrer l'optimalité. En se concentrant sur une sous-partie de l'espace des solutions, cette méthode évite le problème des symétries ce qui améliore sa convergence. Cette hypothèse est appuyée par les résultats d'une étude préliminaire sur le problème de flot insécable dynamique où un seul pas de temps est considéré. En effet, à cause de la présence d'un chemin privilégié pour chaque commodité, cette variante ne possède pas autant de symétries. On constate dans ce cadre une différence moins importante entre les méthodes basées sur les deux normalisations.

Comparaison entre DW-point-intérieur et DW-Fenchel-itératif. Dans le cas du jeu de données *faible demande maximale* où le nombre de commodités est élevé, la méthode DW-Fenchel-itératif se comporte beaucoup mieux que les méthodes de Dantzig-Wolfe. Nous supposons que ceci est dû au fait que le grand nombre de commodités implique une plus grande dégénérescence du problème maître qui est mieux traitée par les méthodes de Fenchel. En effet, cette dégénérescence semble être la cause du démarrage assez lent des méthodes de Dantzig-Wolfe sur ces instances. En revanche, pour le jeu de données *forte demande maximale*, les méthodes DW-point-intérieur et DW-Fenchel-itératif présentent des résultats plus similaires. Sur ces instances, la méthode DW-Fenchel-itératif présente un début de convergence plus rapide, mais ralenti en fin en convergence, en particulier pour la borne inférieure.

Impact de la taille des capacités. Le problème de sac à dos est connu pour posséder des algorithmes de résolutions pseudo-polynomiaux en la capacité du sac à dos tel que l'algorithme MINKNAP que nous utilisons. Dans le cas des flots insécables, cette capacité du sac à dos correspond à la capacité des arcs. Dans les Figures 5.12 et 5.13, nous faisons varier les capacités des arcs. Notons que les résultats pour des capacités de 1000 sont donnés en Figure 5.7. Cette variation des capacités impacte en effet le temps de calcul des deux méthodes qui deviennent plus lentes. Cependant, la méthode DW-Fenchel-itératif est beaucoup impactée. Ceci est dû au fait que cette variation des capacités impacte principalement le sous-problème et en particulier le temps de calcul de l'algorithme MINKNAP. Or, les méthodes ayant un sous-problème de Fenchel passent beaucoup plus de temps dans leur sous-problème que les méthodes de Dantzig-Wolfe et sont donc plus impactées.

Commentaires généraux. Les nouvelles méthodes présentées, couplant les décompositions de Dantzig-Wolfe et de Fenchel, affichent des résultats très prometteurs. En particulier, elles semblent moins affectées par la dégénérescence que la décomposition de Dantzig-Wolfe et possède une meilleure convergence que la décomposition de Fenchel. En revanche, elles peuvent finir de converger moins vite que la décomposition de Dantzig-Wolfe sur les instances où la dégénérescence est moins importante. Par ailleurs, ces méthodes sont particulièrement efficaces lorsque l'oracle d'optimisation (O) peut être implémenté par un algorithme rapide. De plus, on remarque que les techniques de "Réduction de la dimensionnalité" et de "*Lifting*" ont un impact majeur sur le temps de résolution du sous-problème de Fenchel.

5.8 Conclusion

Dans ce chapitre, nous nous sommes intéressés à des méthodes de décomposition capables d'améliorer la relaxation linéaire des problèmes de programmation linéaire en nombres entiers et en particulier des problèmes de flot insécable. Nous avons passé en revue deux méthodes de la littérature, les décompositions de Dantzig-Wolfe et de Fenchel, et avons proposé une nouvelle méthode de décomposition inspirée des deux précédentes. Cette nouvelle méthode utilise les problèmes maîtres de Fenchel et de Dantzig-Wolfe qui sont couplés à travers un sous-problème de Fenchel utilisant une normalisation directionnelle. De plus, nous avons proposé une nouvelle approche de résolution pour le sous-problème de la décomposition de Fenchel dans le cadre spécifique de la normalisation directionnelle. Des garanties théoriques de terminaison ont pu être démontrées pour cette approche et nous avons pu constater expérimentalement qu'elle présente moins de problèmes d'instabilités numériques que l'approche classique. La nouvelle méthode de décomposition proposée a été comparée expérimentalement aux méthodes de la littérature et plusieurs points clé de ces méthodes ont été mis en évidence. Dans cette étude expérimentale, la nouvelle méthode a démontré de très bon résultats. En particulier, bien que la méthode ralentisse parfois en fin de convergence, elle a pu rapidement trouver des bornes de bonne qualité et ce même pour des instances contenant des graphes de 400 nœuds.

Une piste probable pour de futures recherches sera donc d'investiguer les performances de cette nouvelle méthode dans un contexte différent de celui des problèmes de flot insécable. De plus, l'un des points centraux de cette nouvelle méthode est l'utilisation d'une normalisation directionnelle dans le sous-problème de Fenchel. Il serait intéressant d'utiliser cette normalisation à l'intérieur d'autres méthodes de décomposition.

Conclusion

Dans cette thèse, nous avons étudié le problème de la transmission de ressources indivisibles au travers d'un réseau. Ce problème possède de nombreuses applications dans le transport de fret et dans les télécommunications (réseaux optiques, communications satellitaires ...). En particulier, l'application à l'origine de cette thèse est la constellation de satellites Telesat. L'industrie des constellations de télécommunication construit des constellations contenant de plus en plus de satellites afin de servir de plus en plus d'utilisateurs. Ceci tend à créer des problèmes de transmission de ressources de plus en plus difficiles à résoudre ce qui nécessite des algorithmes de résolution plus performants. Ainsi, afin de résoudre ce problème, nous avons commencé par modéliser mathématiquement le problème de transmission dans la constellation Telesat. En nous restreignant à une sous-partie de ce problème, nous en sommes venus à étudier des problèmes d'optimisation connus de la littérature : les problèmes de flot insécable statiques et dynamiques.

Pour le problème statique, nous avons présenté une heuristique basée sur l'arrondi aléatoire étendant l'algorithme de Raghavan et Tompson (1987). Nous avons montré expérimentalement que sur des instances de grande taille, cette heuristique produit des solutions de meilleure qualité que les autres méthodes avec laquelle elle a été comparée. En particulier, elle est capable de résoudre des instances ayant plus de 400 nœuds ce qui est l'ordre de grandeur des instances de notre application industrielle. De plus, nous avons créé une variante de l'heuristique ayant des propriétés d'approximation et le facteur d'approximation de cette variante et de l'algorithme de Raghavan et Tompson (1987) a été amélioré pour dépendre d'un paramètre de granularité de la demande. Ce paramètre permet de comprendre le comportement de l'arrondi aléatoire lorsque les demandes des commodités sont petites par rapport aux capacités des liens de transmission. De plus, le comportement de l'heuristique présentée a été analysé pour mettre en évidence deux de ses particularités clé. Ainsi, nous avons pu discuter de l'impact de la fonction objectif et de l'actualisation de la solution de la relaxation linéaire sur les performances de l'heuristique.

L'une des limitations du problème de flot insécable statique est qu'il ne prend pas en compte la dynamique de la constellation, car il ne la considère qu'à un instant fixé. C'est pourquoi nous avons étudié le problème de flot insécable dynamique qui permet de prendre en compte cet aspect dans la modélisation. Plusieurs nouvelles méthodes capables de résoudre des instances de moyenne et grande taille du problème de flot insécable dynamique ont été présentées. En particulier, de nouvelles formulations ont été introduites qui modélisent soit le problème en nombres entiers, soit sa relaxation linéaire. De plus, nous avons introduit de nouvelles approches de résolution pour le problème de *pricing* utilisé dans la formulation introduite par Gamvros et Raghavan (2012). Ces méthodes ne reposent pas sur des calculs de k-plus-courts chemins et ne partagent donc pas les limitations de l'approche précédente : le nombre de k-plus-courts chemins à calculer peut être exponentiel en la taille du graphe considéré. Les formulations pour le problème dynamique ont été intégrées dans des solveurs matheuristiques qui viennent compléter l'approche de résolution exacte proposée par Gamvros

et Raghavan (2012). Ces solveurs ont été comparé expérimentalement et plusieurs de leurs aspects clé expliquant leurs performances ont été mis en évidence.

La plupart des méthodes de résolution utilisées pour les problèmes de flot insécable statiques ou dynamiques reposent sur des calculs de relaxation linéaire. Afin de les améliorer, nous nous sommes intéressés à des méthodes de décomposition capables d'améliorer la relaxation linéaire des problèmes de programmation linéaire en nombres entiers et en particulier des problèmes de flot insécable. Nous avons passé en revue deux méthodes de la littérature, les décompositions de Dantzig-Wolfe et de Fenchel, et avons proposé une nouvelle méthode de décomposition inspirée des deux précédentes. De plus, nous avons proposé une nouvelle approche de résolution pour le sous-problème de la décomposition de Fenchel. Des garanties théoriques de terminaison ont pu être démontrées pour cette approche et nous avons pu constater expérimentalement qu'elle présente moins de problèmes d'instabilités numériques que l'approche classique. La nouvelle méthode de décomposition proposée a été comparée expérimentalement aux méthodes de la littérature et plusieurs points clé de ces méthodes ont été mis en évidence. Dans cette étude expérimentale, la nouvelle méthode a démontré des résultats très prometteurs.

Comme souligné en conclusion des chapitres, plusieurs perspectives sont envisageables pour continuer les travaux de cette thèse. Premièrement, les ajouts différenciant l'heuristique SRR de l'algorithme de Raghavan et Tompson (1987) s'appliquent à d'autres contextes que les flots insécables. Ils pourraient, par exemple, être étudiés dans le contexte des problèmes de recouvrement ou de *packing*. Deuxièmement, il serait intéressant de considérer une remise en cause de certaines des décisions prises lors des algorithmes d'arrondi aléatoire ; par exemple en utilisant des méthodes de *backtracking*. Troisièmement, bien que les métaheuristiques n'aient pas été étudiées dans le cadre du problème de flot insécable dynamique, elles pourraient être une alternative aux méthodes basées sur des formulations de programmation linéaire (en nombres entiers). Une autre piste de futures recherches serait d'investiguer les performances de la méthode de décomposition présentée dans le Chapitre 5 dans un contexte différent de celui des problèmes de flot insécable. Enfin, l'un des points centraux de cette nouvelle méthode de décomposition est l'utilisation d'une normalisation directionnelle dans le sous-problème de Fenchel. Il serait intéressant de considérer l'utilisation de cette normalisation à l'intérieur d'autres méthodes de décomposition.

Par ailleurs, nous n'avons considéré dans cette thèse qu'une partie du problème de gestion des ressources de télécommunication dans une constellation de satellites. En effet, une des hypothèses faites au début de cette thèse était que les liens utilisés pour transmettre le débit internet entre les utilisateurs et les satellites étaient déjà choisis et donnés en entrée de nos algorithmes. Or, effectuer ce choix est un problème d'optimisation complexe qui mérite lui aussi d'être étudié en détails. De plus, pour trouver la meilleure solution possible au problème de gestion des constellations, ce problème doit être résolu en même temps que le problème de transmission que nous avons étudié dans cette thèse. Faire communiquer ces deux problèmes ou les unifier en un seul pouvant être résolu par un unique algorithme est donc une piste de recherche. Enfin, notre modélisation du problème de gestion des constellations considère uniquement les aspects déterministes des constellations. Or, durant la vie d'une constellation,

de nombreux événements aléatoires peuvent survenir tels que des événements météorologiques ou des défaillances du matériel. De plus, les utilisateurs auront une demande stochastique à laquelle il conviendra de s'ajuster. Inclure ses éléments dans la modélisation demandera de grandement modifier les algorithmes utilisés pour gérer les ressources de télécommunication d'une constellation de satellites telle que Telesat.

Annexe A

Méthode de la génération de colonnes

A.1 Cas général

La génération de colonnes est une technique de programmation linéaire destinée à résoudre les modèles possédant un très grand nombre de variables. L'idée dans cette méthode est de résoudre le modèle considéré avec un sous-ensemble de ses variables et de rajouter itérativement des variables au modèle jusqu'à être capable de démontrer qu'ajouter de nouvelles variables ne permettrait plus d'améliorer la valeur de la fonction objectif. Le point difficile de cette méthode est de déterminer quelles sont les variables à ajouter au modèle. C'est ce que nous détaillons dans les paragraphes suivants. Considérons le programme linéaire sous forme standard suivant :

$$\min_x c^T x \tag{A.1a}$$

tel que

$$Ax = b \tag{A.1b}$$

$$x \in \mathbb{R}^+ \tag{A.1c}$$

que nous nommerons problème primal ainsi que son programme linéaire dual :

$$\max_u u^T b \tag{A.2a}$$

tel que

$$u^T A \leq c \tag{A.2b}$$

$$u \in \mathbb{R} \tag{A.2c}$$

De plus, soit x^* et u^* des solutions optimales pour ces deux problèmes pouvant être fournies par n'importe quel solveur linéaire. Ces solution vérifient les contraintes de leur programme linéaire et possède la même valeur de fonction objectif ($c^T x^* = u^{*T} b$) que nous

nommerons z^*. Cette valeur optimale est une fonction des différents coefficients du problème primal : $z^* = z^*(c, A, b)$. Notons qu'il existe une variable duale u_i^* pour chaque contrainte du modèle linéaire primal. Il est possible de montrer qu'une variable duale optimale u_i^* peut être interprétée comme la dérivé partielle de la valeur optimale z^* de la fonction objectif par rapport au coefficient b_i du second membre des contraintes : $u_i^* = \frac{\partial z^*}{\partial b_i}$ ou encore $u^* = \frac{\partial z^*}{\partial b}$. Plus simplement, u_i^* indique de combien augmente localement la valeur optimale de la fonction objectif lorsque le coefficient b_i augmente d'une unité.

Considérons maintenant qu'une variable y n'était pas considérée jusque là dans le problème primal. Ceci est équivalant à dire que la variable y était présente dans le modèle mais prenait une valeur nulle. On va maintenant observer l'impact sur le problème primal du changement de valeur de y qui sera maintenant fixée à $\hat{y}$. Si c_y et A_y sont respectivement les coefficients associés à la variable y dans la fonction objectif et dans les contraintes alors le programme linéaire est modifié de la manière suivante :

$$\min_x c^T x + c_y \hat{y} \tag{A.3a}$$

tel que

$$Ax = b - A_y \hat{y} \tag{A.3b}$$

$$x \in \mathbb{R}^+ \tag{A.3c}$$

Afin de savoir s'il est intéressant de rajouter la variable y au problème (*i.e.* de lui laisser prendre une valeur non nulle), on veut savoir si la valeur $z_{\hat{y}}^*$ de la fonction objectif de ce nouveau problème diminue lorsque la valeur $\hat{y}$ de la variable y augmente. Autrement dit, on veut connaître $\frac{dz_{\hat{y}}^*}{d\hat{y}}$. Pour ce faire, remarquons que $z_{\hat{y}}^*$ peut s'exprimer en fonction de la valeur de la fonction objectif du problème primal initial : $z_{\hat{y}}^* = c_y \hat{y} + z^*(c, A, b - A_y \hat{y})$. On peut alors calculer la dérivé que nous intéresse :

$$\frac{dz_{\hat{y}}^*}{d\hat{y}} = \frac{\partial z_{\hat{y}}^*}{\partial \hat{y}} + \frac{dz^*}{d\hat{y}} \tag{A.4a}$$

$$= c_y + \frac{\partial z^*}{\partial c}\frac{dc}{d\hat{y}} + \frac{\partial z^*}{\partial A}\frac{dA}{d\hat{y}} + \frac{\partial z^*}{\partial b}\frac{db}{d\hat{y}} \tag{A.4b}$$

$$= c_y + \frac{\partial z^*}{\partial b}\frac{db}{d\hat{y}} \tag{A.4c}$$

$$= c_y + u^*(-A_y) \tag{A.4d}$$

$$= c_y - u^* A_y \tag{A.4e}$$

Autrement dit, l'impact du changement de la valeur $\hat{y}$ sur la valeur $z_{\hat{y}}^*$ se traduit en deux termes. Premièrement, ce changement impact directement la fonction objectif et deuxièmement, le second membre des contraintes est modifié ce qui a un impact sur les variables

optimales x^* dont l'ampleur est mesurée à l'aide des variables duales u^*. La dérivée $\frac{dz^*_{\hat{y}}}{d\hat{y}}$ est généralement appelée coût réduit de la variable y et sera notée cr_y dans la suite.

La génération de colonnes possède la même condition d'arrêt que la méthode du simplexe : la solution courante est optimale si et seulement si toutes les variables ont un coût réduit positif, *i.e.* $\forall y, cr_y \geq 0$. En effet, cette condition et la convexité des problèmes d'optimisation linéaire indiquent que, pour toutes les variables non considérées dans le problème (*i.e.* fixées à zéro), augmenter la valeur de ces variables augmente la valeur de la fonction objectif. Dans ce cas, on est assuré qu'ajouter de nouvelles variables au problème ne permettra pas d'obtenir une meilleure solution. Une itération de la génération de colonnes procède donc de la manière suivante :

1. Initialiser le problème avec un sous-ensemble de ses variables.
2. Calculer une solution primale-duale (x^*, u^*) du problème maître.
3. A l'aide de u^*, chercher une variable de coût réduit négatif.
4. S'il n'existe pas de telle variable, la méthode s'arrête : (x^*, u^*) est la solution optimale du problème.
5. Sinon, ajouter la variable de coût réduit négatif au problème maître et reprendre à l'étape 2.

Lorsque le nombre de variables est grand, la vérification des conditions d'optimalité en calculant tous les coûts réduits n'est pas envisageable. Cependant, les conditions d'optimalité sont équivalentes à ce que le coût réduit minimum parmi l'ensemble des variables soit positif, c'est-à-dire $\min_y cr_y \geq 0$. L'idée est alors de ne calculer que la variable de coût réduit minimum à l'aide d'un problème d'optimisation appelé sous-problème de *pricing*. Ce sous-problème dépend fortement de la structure du problème initial. La méthode de la génération de colonnes est particulièrement performante lorsque cette structure permet de résoudre le sous-problème avec un algorithme efficace, typiquement un algorithme combinatoire dédié. Dans les prochaines sections, nous présentons une application de la génération aux deux problèmes étudiés dans cette thèse, les problèmes de flot insécable statique et dynamique.

A.2 Application au problème de flot insécable statique

Dans cette section, nous appliquons la génération de colonnes à la relaxation du problème de flot insécable. Considérons la relaxation linéaire de la formulation par chemins présentée

en Section 2.1.3 :

$$\min_{x_p^k, o_e} \sum_{e \in E} o_e \tag{A.5a}$$

tel que

$$\sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K, \tag{A.5b}$$

$$\sum_{k \in K} \sum_{p \in P^k | e \in p} x_p^k d^k \leq c_e + o_e \qquad \forall e \in E, \tag{A.5c}$$

$$x_p^k \in \mathbb{R}^+,\ o_e \in \mathbb{R}^+ \qquad \forall p \in \bigcup_k P^k,\ \forall k \in K,\ \forall e \in E. \tag{A.5d}$$

Cette formulation possède une variable x_p^k pour chaque chemin valide et pour chaque commodité. Cela représente un nombre exponentiel de variables en le nombre d'arcs et de nœuds. Cette formulation est donc généralement résolue à l'aide d'une génération de colonnes. Pour ce faire, un unique chemin est initialement considéré pour chaque commodité : typiquement le plus court entre l'origine et la destination de chaque commodité. Nous détaillons maintenant le calcul de la variable de coût réduit minimum car c'est le point central de la génération de colonnes.

Le coût réduit d'une variable dépend de ses coefficients dans la fonction objectif et dans chacune des contraintes ainsi que des variables duales associées à chacune des contraintes. Dans le cas de la formulation précédente, une variable x_p^k a :

- un coefficient nul dans la fonction objectif ;
- un coefficient unitaire dans la contrainte (A.5b) associé à sa commodité ;
- un coefficient d^k dans les contraintes de capacité (A.5c) associées aux arcs utilisés par le chemin p.

Si on note respectivement u^k et u_e les variables duales associées aux contraintes (A.5b) et (A.5c), le coût réduit d'une variable x_p^k est $-u^k - d^k \sum_{e \in p} u_e$. Afin de calculer la variable de coût réduit minimum, un problème de *pricing* est résolu pour chaque commodité indépendamment. Pour une commodité fixée, on constate que le terme additif $-u^k$ et le terme multiplicatif d^k sont constants. Ils ne sont donc pas considérés directement dans le problème d'optimisation du coût réduit. Le problème de *pricing* pour une commodité est donc de trouver un chemin minimisant $-\sum_{e \in p} u_e$: c'est un problème de plus court chemin au sens de l'opposé des variables duales. Puisque les variables duales u_e sont toujours négatives (car les contraintes associées sont $\leq$ dans un problème de minimisation), cette minimisation peut être effectuée très rapidement à l'aide de l'algorithme de Dijkstra. Une fois que le meilleur chemin est calculé pour chaque commodité, la génération de colonnes indique qu'il faut ajouter au problème maître la variable associée au meilleur d'entre eux si son coût réduit est négatif. Cependant, il est plus efficace en pratique d'ajouter toutes les variables de coût réduit négatif calculées.

Annexe B

Figures complémentaires pour les résultats expérimentaux sur le problème de flot insécable dynamique

Nous donnons dans les Figures B.2, B.3 et B.4 des résultats de test supplémentaires pour le problème de flot insécable dynamique. En effet, pour la comparaison des algorithmes pénalisés et non pénalisés nous rajoutons les résultats pour les jeux de données *grille difficile*, *aléatoire connexe* et *taille des commodités*. De plus, pour la comparaison des ordres d'arrondi, nous rajoutons les résultats pour les jeux de données *grille facile*, *grille difficile* et *aléatoire connexe*. Ces résultats sont commentés en Section 4.3.5 et sont donné ici par souci d'exhaustivité.

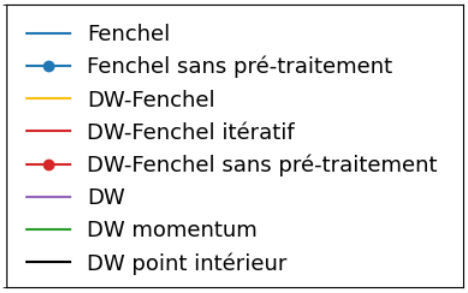
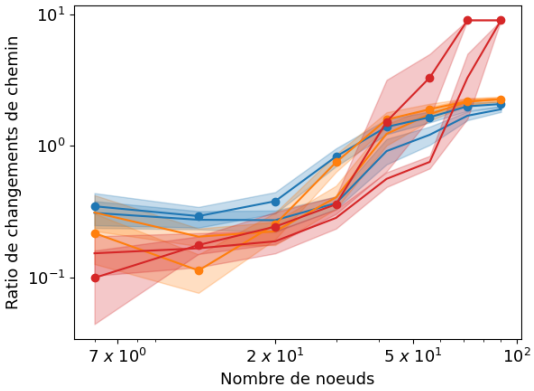
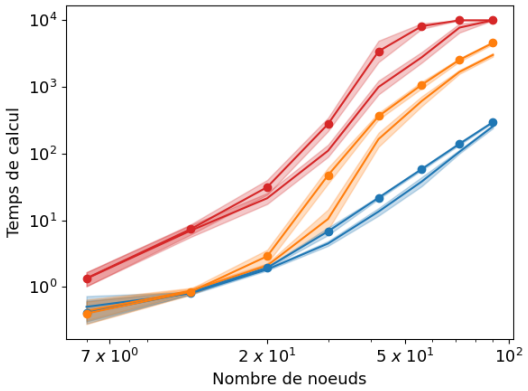


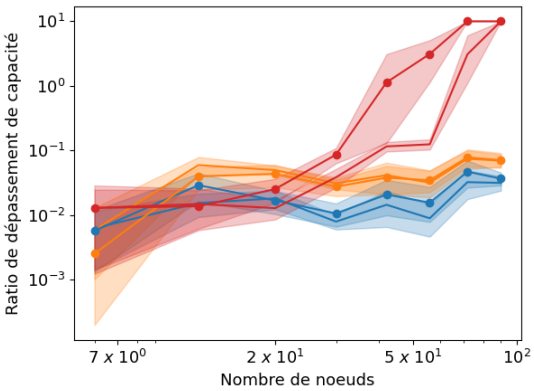
Figure B.1 – Légende des Figures B.2 à B.4



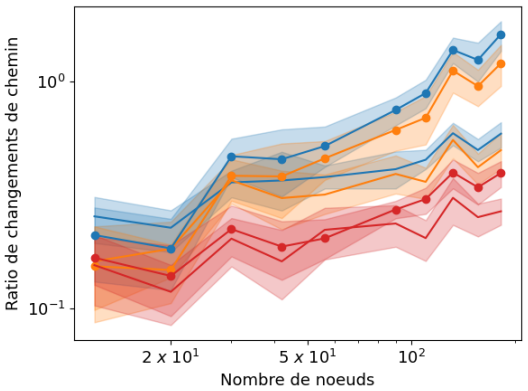
(a) Comparaison pénalisation, ratio de changements de chemin, jeu de données *grille difficile*



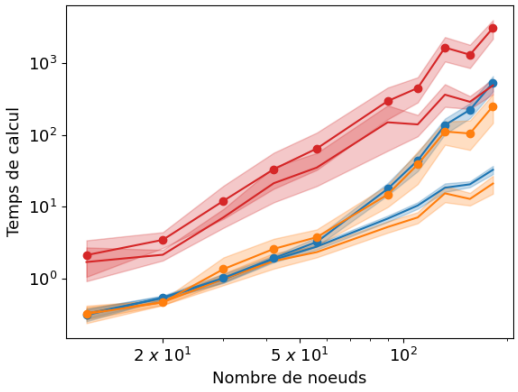
(b) Comparaison pénalisation, temps de calcul, jeu de données *grille difficile*



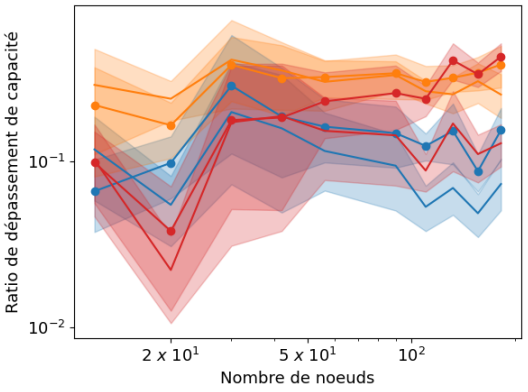
(c) Comparaison pénalisation, ratio de dépassement, jeu de données *grille difficile*



(d) Comparaison pénalisation, ratio de changements de chemin, jeu de données *aléatoire connexe*

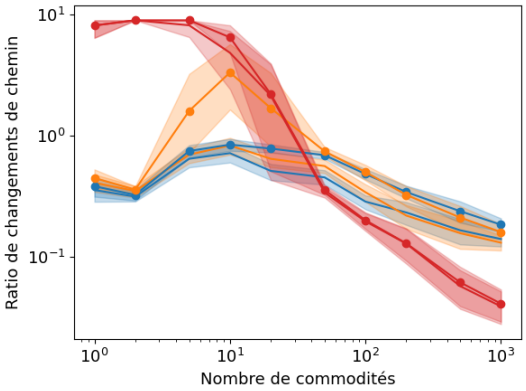


(e) Comparaison pénalisation, temps de calcul, jeu de données *aléatoire connexe*

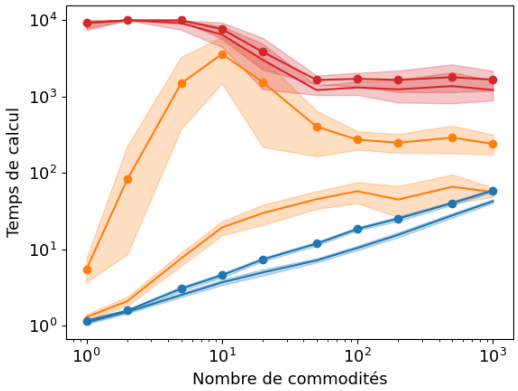


(f) Comparaison pénalisation, ratio de dépassement, jeu de données *aléatoire connexe*

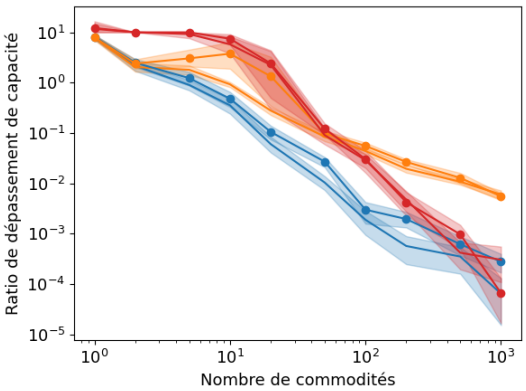
FIGURE B.2 – Comparaison des algorithmes pénalisés et non pénalisés sur les jeux de données *grille difficile* et *aléatoire connexe*



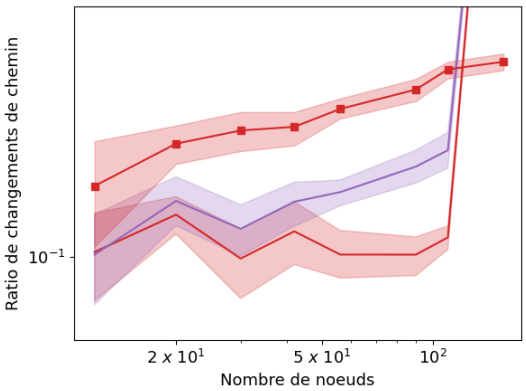
(a) Comparaison pénalisation, ratio de changements de chemin, *taille des commodités* dataset



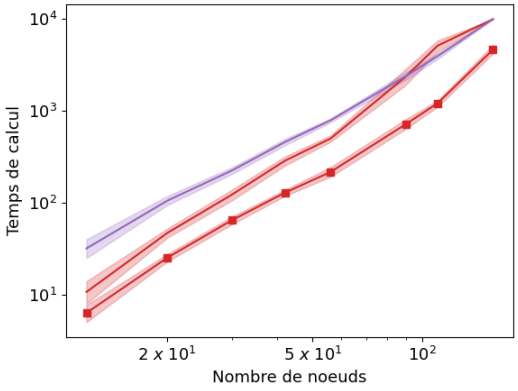
(b) Comparaison pénalisation, temps de calcul, *taille des commodités* dataset



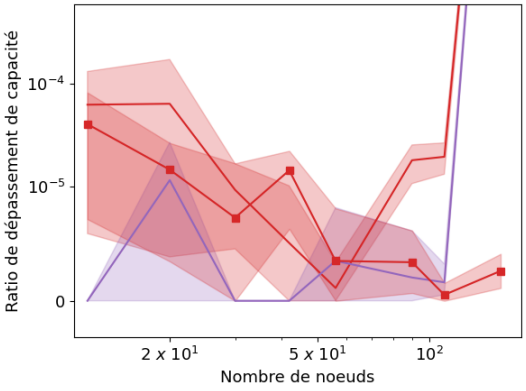
(c) Comparaison pénalisation, ratio de dépassement, *taille des commodités* dataset



(d) Ordre d'arrondi, ratio de changements de chemin, jeu de données *grille facile*

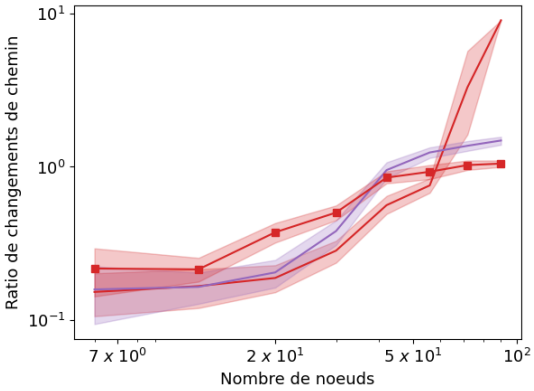


(e) Ordre d'arrondi, temps de calcul, *grille facile* dataset

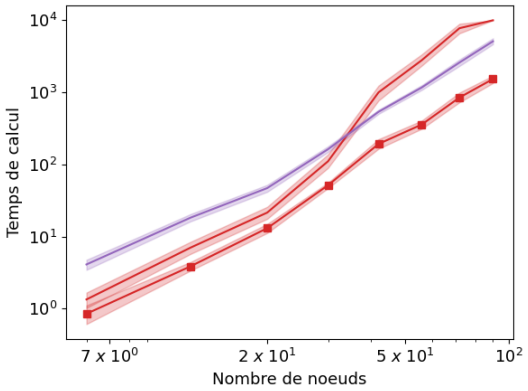


(f) Ordre d'arrondi, ratio de dépassement, jeu de données *grille facile*

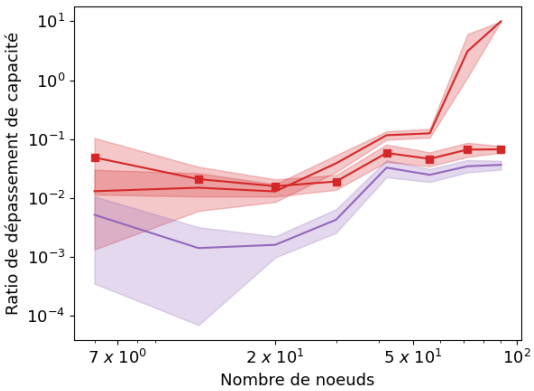
FIGURE B.3 – Comparaison des algorithmes pénalisés et non pénalisés sur le jeu de données *taille des commodités* et comparaison de différents ordres d'arrondi sur le jeu de données *grille facile*



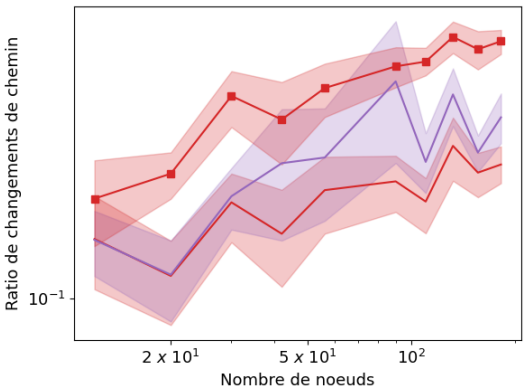
(a) Ordre d'arrondi, ratio de changements de chemin, jeu de données *grille difficile*



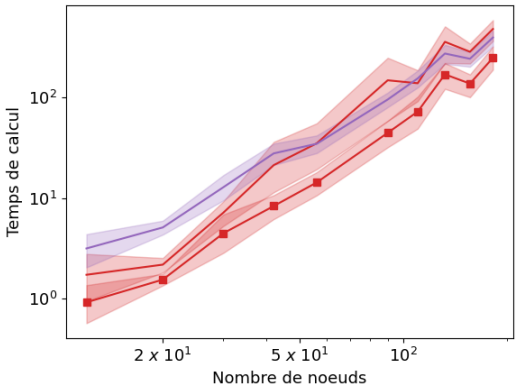
(b) Ordre d'arrondi, temps de calcul, jeu de données *grille difficile*



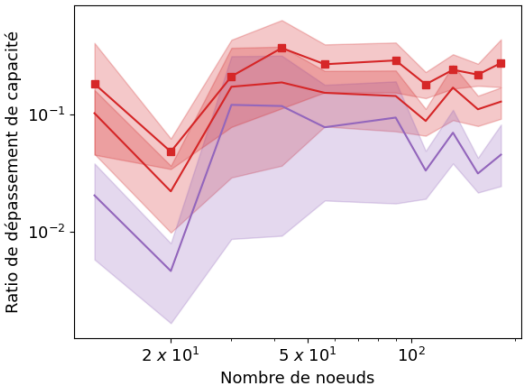
(c) Ordre d'arrondi, ratio de dépassement, jeu de données *grille difficile*



(d) Ordre d'arrondi, ratio de changements de chemin, jeu de données *aléatoire connexe*



(e) Ordre d'arrondi, temps de calcul, jeu de données *aléatoire connexe*



(f) Ordre d'arrondi, ratio de dépassement, jeu de données *aléatoire connexe*

FIGURE B.4 – Comparaison de différents ordres d'arrondi sur les jeux de données *grille difficile* et *aléatoire connexe*

Bibliographie

Filipe Alvelos et JM Valério De Carvalho : Comparing branch-and-price algorithms for the unsplittable multicommodity flow problem. *In International Network Optimization Conference*, pages 7–12, 2003.

Filipe Alvelos et Jose Manuel Valério de Carvalho : A local search heuristic based on column generation applied to the binary multicommodity flow problem. *In Proceedings of International Network Optimization Conference, INOC*, page 6, 2007.

Charles A Anderson, Kathryn Fraughnaugh, Mark Parker et Jennifer Ryan : Path assignment for call routing : An application of tabu search. *Annals of Operations Research*, 41(4):299–312, 1993.

Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar et Lisa Zhang : Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

Yasuhito Asano : Experimental evaluation of approximation algorithms for the minimum cost multiple-source unsplittable flow problem. *In ICALP Satellite Workshops*, pages 111–122, 2000.

Alper Atamtürk et Deepak Rajan : On splittable and unsplittable flow capacitated network design arc–set polyhedra. *Mathematical Programming*, 92(2):315–333, 2002.

Yonatan Aumann et Yuval Rabani : Improved bounds for all optical routing. *In Proceedings of the sixth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 567–576, 1995.

Pasquale Avella, Maurizio Boccia et Igor Vasilyev : A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45(3):543–555, 2010.

Yossi Azar et Oded Regev : Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006.

Frédéric Babonneau, Olivier Du Merle et Jean-Phillipe Vial : Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. *Operations Research*, 54(1):184–197, 2006.

Georg Baier, Ekkehard Köhler et Martin Skutella : On the k-splittable flow problem. *In European Symposium on Algorithms*, pages 101–113. Springer, 2002.

Egon Balas et Michael Perregaard : Lift-and-project for mixed 0–1 programming : recent progress. *Discrete Applied Mathematics*, 123(1-3):129–154, 2002.

Cynthia Barnhart, Christopher A Hane et Pamela H Vance : Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

Pierre-Olivier Bauguion, Walid Ben-Ameur et Eric Gourdin : A new model for multi-commodity flow problems, and a strongly polynomial algorithm for single-source maximum concurrent flow. *Electronic Notes in Discrete Mathematics*, 41:311–318, 2013.

Pierre-Olivier Bauguion, Walid Ben-Ameur et Eric Gourdin : Efficient algorithms for the maximum concurrent flow problem. *Networks*, 65(1):56–67, 2015.

Eric Beier, Saravanan Venkatachalam, Luca Corolli et Lewis Ntaimo : Stage-and scenario-wise fenchel decomposition for stochastic mixed 0-1 programs with special structure. *Computers & Operations Research*, 59:94–103, 2015.

Meriema Belaidouni et Walid Ben-Ameur : On the minimum cost multiple-source unsplittable flow problem. *RAIRO-Operations Research*, 41(3):253–273, 2007.

Walid Ben-Ameur et José Neto : Acceleration of cutting-plane and column generation algorithms : Applications to network design. *Networks : An International Journal*, 49 (1):3–17, 2007.

Amal Benhamiche, A Ridha Mahjoub, Nancy Perrot et Eduardo Uchoa : Unsplittable non-additive capacitated network design using set functions polyhedra. *Computers & Operations Research*, 66:105–115, 2016.

Amal Benhamiche, A Ridha Mahjoub, Nancy Perrot et Eduardo Uchoa : Capacitated multi-layer network design with unsplittable demands : polyhedra and branch-and-cut. *Discrete Optimization*, 35:100555, 2020.

Paweł M Białoń : A randomized rounding approach to a k-splittable multicommodity flow problem with lower path flow bounds affording solution quality guarantees. *Telecommunication Systems*, 64(3):525–542, 2017.

Maurizio Boccia, Antonio Sforza, Claudio Sterle et Igor Vasilyev : A cut and branch approach for the capacitated p-median problem based on fenchel cutting planes. *Journal of Mathematical Modelling and Algorithms*, 7(1):43–58, 2008.

Pierre Bonami : *Etude et mise en œuvre d'approches polyédriques pour la résolution de programmes en nombres entiers ou mixtes généraux*. Thèse de doctorat, Paris 6, 2003.

E Andrew Boyd : Generating fenchel cutting planes for knapsack polyhedra. *SIAM Journal on Optimization*, 3(4):734–750, 1993.

E Andrew Boyd : Fenchel cutting planes for integer programs. *Operations Research*, 42 (1):53–64, 1994.

E Andrew Boyd : On the convergence of fenchel cutting planes in mixed-integer programming. *SIAM Journal on Optimization*, 5(2):421–435, 1995.

Olivier Briant, Claude Lemaréchal, Phillipe Meurdesoif, Sophie Michel, Nancy Perrot et François Vanderbeck : Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.

Marcel Büther et Dirk Briskorn : Reducing the 0-1 knapsack problem with a single continuous variable to the standard 0-1 knapsack problem. *International Journal of Operations Research and Information Systems (IJORIS)*, 3(1):1–12, 2012.

Massimiliano Caramia et Antonino Sgalambro : An exact approach for the maximum concurrent k-splittable flow problem. *Optimization Letters*, 2(2):251–265, 2008.

Massimiliano Caramia et Antonino Sgalambro : A fast heuristic algorithm for the maximum concurrent k-splittable flow problem. *Optimization Letters*, 4(1):37–55, 2010.

Jordi Castro et Jordi Cuesta : Improving an interior-point algorithm for multicommodity flows by quadratic regularizations. *Networks*, 59(1):117–131, 2012.

Amit Chakrabarti, Chandra Chekuri, Anupam Gupta et Amit Kumar : Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.

Liang Chen, Wei-Kun Chen, Mu-Ming Yang et Yu-Hong Dai : An exact separation algorithm for unsplittable flow capacitated network design arc-set polyhedron. *Journal of Global Optimization*, pages 1–31, 2021.

Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna et Kunal Talwar : Hardness of routing with congestion in directed graphs. *In Proceedings of the thirty-ninth annual ACM Symposium on Theory of Computing*, pages 165–178. ACM, 2007.

Vašek Chvátal, William Cook et Daniel Espinoza : Local cuts for mixed-integer programming. *Mathematical Programming Computation*, 5(2):171–200, 2013.

Michele Conforti et Laurence A Wolsey : "facet" separation with one linear program. *Mathematical Programming*, 178(1):361–380, 2019.

Ivan Contreras, Jean-François Cordeau et Gilbert Laporte : The dynamic uncapacitated hub location problem. *Transportation Science*, 45(1):18–32, 2011.

D. Coudert et H. Rivano : Lightpath assignment for multifibers wdm networks with wavelength translators. *In Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, volume 3, pages 2686–2690 vol.3, 2002.

Louis Anthony Cox : Dynamic anticipatory routing of circuit-switched telecommunications networks. *Handbook of Genetic Algorithms*, 1991.

János Csirik, G Galambos, Hans JBG Frenk, AM Frieze, Rinnooy Kan et AHG Alexander : A probabilistic analysis of the next fit decreasing bin packing heuristic. *Operations Research Letters*, 5(5):233–236, 1986.

Weibin Dai, Xiaoqian Sun et Sebastian Wandelt : Finding feasible solutions for multicommodity flow problems. *In 2016 35th Chinese Control Conference (CCC)*, pages 2878–2883. IEEE, 2016a.

Weibin Dai, Jun Zhang et Xiaoqian Sun : On solving multi-commodity flow problems : An experimental evaluation. *Chinese Journal of Aeronautics*, 30(4):1481–1492, 2017.

Weibin Dai, Jun Zhang, Xiaoqian Sun et Sebastian Wandelt : Node dependency in multi-commodity flow problem with applications to transportation networks. *In 16th COTA International Conference of Transportation Professionals*, pages 1989–2001, 2016b.

Sanjeeb Dash : Mixed integer rounding cuts and master group polyhedra. *In Combinatorial Optimization*, volume 31, pages 1–32. IOS Press, 2011.

Guy Desaulniers, Jacques Desrosiers et Marius M Solomon : *Column generation*, volume 5. Springer Science & Business Media, 2006.

Yefim Dinitz, Naveen Garg et Michel X Goemans : On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.

Bradley Efron : Bootstrap methods : another look at the jackknife. *In Breakthroughs in statistics*, pages 569–593. Springer, 1992.

Kristoffer Emanuelsson : Approximating multi-commodity max-flow in practice. Mémoire de D.E.A., KTH Royal Institute of Technology, 2016.

Thomas Erlebach et Alexander Hall : Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *Journal of Scheduling*, 7(3):223–241, 2004.

Lisa K Fleischer : Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

Lester R Ford Jr : Network flow theory. Rapport technique, Rand Corp Santa Monica Ca, 1956.

Ioannis Fragkos, Jean-François Cordeau et Raf Jans : *The multi-period multi-commodity network design problem.* CIRRELT, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2017.

Ricardo Fukasawa et Marcos Goycoolea : On the exact separation of mixed integer knapsack cuts. *Mathematical Programming*, 128(1):19–41, 2011.

Mette Gamst : A decomposition based on path sets for the multi-commodity k-splittable maximum flow problem. *DTU Management Engineering*, 2013.

Mette Gamst : A local search heuristic for the multi-commodity k-splittable maximum flow problem. *Optimization Letters*, 8(3):919–937, 2014.

Mette Gamst, Peter Neergaard Jensen, David Pisinger et Christian Plum : Two-and three-index formulations of the minimum cost multicommodity k-splittable flow problem. *European Journal of Operational Research*, 202(1):82–89, 2010.

Mette Gamst et Bjørn Petersen : Comparing branch-and-price algorithms for the multi-commodity k-splittable maximum flow problem. *European Journal of Operational Research*, 217(2):278–286, 2012.

Ioannis Gamvros et Subramanian Raghavan : Multi-period traffic routing in satellite networks. *European Journal of Operational Research*, 219(3):738–750, 2012.

Jean-Louis Goffin et Jean-Philippe Vial : Convex nondifferentiable optimization : A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17(5):805–867, 2002.

Jacek Gondzio et Pablo González-Brevis : A new warmstarting strategy for the primal-dual column generation method. *Mathematical Programming*, 152(1-2):113–146, 2015.

Jacek Gondzio, Pablo González-Brevis et Pedro Munari : New developments in the primal–dual column generation technique. *European Journal of Operational Research*, 224 (1):41–51, 2013.

Jacek Gondzio, Pablo González-Brevis et Pedro Munari : Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation*, 8 (1):47–82, 2016.

Teofilo F. Gonzalez : *Handbook of Approximation Algorithms and Metaheuristics, Second Edition : Two-Volume Set*. Taylor & Francis Group, 2020. ISBN 9780367570286.

LLC Gurobi Optimization : Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Yichao He, Jinghong Wang, Xinlu Zhang, Huanzhe Li et Xuejing Liu : Encoding transformation-based differential evolution algorithm for solving knapsack problem with single continuous variable. *Swarm and Evolutionary Computation*, 50:100507, 2019.

Chengwen Jiao, Suixiang Gao et Wenguo Yang : Comparing algorithms for minimizing congestion and cost in the multi-commodity k-splittable flow. *Computer and Information Science*, 8(2):1, 2015.

Chengwen Jiao, Wenguo Yang, Suixiang Gao, Yinben Xia et Mingming Zhu : The k-splittable flow model and a heuristic algorithm for minimizing congestion in the mpls networks. *In 2014 10th International Conference on Natural Computation (ICNC)*, pages 1050–1055. IEEE, 2014.

Víctor M Jiménez et Andrés Marzal : Computing the k-shortest paths : A new algorithm and an experimental comparison. *In International Workshop on Algorithm Engineering*, pages 15–29. Springer, 1999.

Konstantinos Kaparis et Adam N Letchford : Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124(1):69–91, 2010.

Jon M Kleinberg : Single-source unsplittable flow. *In Proceedings of 37th Conference on Foundations of Computer Science*, pages 68–77. IEEE, 1996.

Ronald Koch et Ines Spenke : Complexity and approximability of k-splittable flows. *Theoretical Computer Science*, 369(1-3):338–347, 2006.

Stavros G Kolliopoulos et Clifford Stein : Improved approximation algorithms for unsplittable flow problems. *In Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 426–436. IEEE, 1997.

Petr Kolman : A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*, 88(3):101–105, 2003.

Attila A Kovacs, Bruce L Golden, Richard F Hartl et Sophie N Parragh : Vehicle routing problems in which consistency considerations are important : A survey. *Networks*, 64(3):192–213, 2014.

Manuel Laguna et Fred Glover : Bandwidth packing : a tabu search approach. *Management Science*, 39(4):492–500, 1993.

Francois Lamothe, Emmanuel Rachelson, Alain Haït, Cedric Baudoin et Jean-Baptiste Dupé : The dynamic unsplittable flow problem. En préparation.

François Lamothe, Emmanuel Rachelson, Alain Haït, Cedric Baudoin et Jean-Baptiste Dupé : Randomized rounding algorithms for large scale unsplittable flow problems. *Journal of Heuristics*, 27(6):1081–1110, 2021.

Der-Horng Lee et Meng Dong : Dynamic network design for reverse logistics operations under uncertainty. *Transportation Research part E : logistics and transportation review*, 45 (1):61–71, 2009.

Xian Yong Li, Yash P Aneja et F Baki : An ant colony optimization metaheuristic for single-path multicommodity network flow problems. *Journal of the Operational Research Society*, 61(9):1340–1355, 2010.

Geng Lin, Wenxing Zhu et M Montaz Ali : An exact algorithm for the 0–1 linear knapsack problem with a single continuous variable. *Journal of Global Optimization*, 50(4):657–673, 2011.

Hongtao Liu : An exact algorithm for the biobjective 0-1 linear knapsack problem with a single continuous variable. *In 2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 81–85. IEEE, 2017.

Zhixing Luo, Hu Qin, ChanHou Che et Andrew Lim : On service consistency in multi-period vehicle routing. *European Journal of Operational Research*, 243(3):731–744, 2015.

Aleksander Madry : Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. *In Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130. ACM, 2010.

Hugues Marchand et Laurence A Wolsey : The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85(1):15–33, 1999.

Maren Martens, Fernanda Salazar et Martin Skutella : Convex combinations of single source unsplittable flows. *In European Symposium on Algorithms*, pages 395–406. Springer, 2007.

Maren Martens et Martin Skutella : Flows on few paths : Algorithms and lower bounds. *Networks : An International Journal*, 48(2):68–76, 2006.

Richard Kipp Martin : *Large scale linear and integer optimization : a unified approach*. Springer Science & Business Media, 2012.

Hela Masri, Saoussen Krichen et Adel Guitouni : An ant colony optimization metaheuristic for solving bi-objective multi-sources multicommodity communication flow problem. *In 2011 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011)*, pages 1–8. IEEE, 2011.

Hela Masri, Saoussen Krichen et Adel Guitouni : A multi-start variable neighborhood search for solving the single path multicommodity flow problem. *Applied Mathematics and Computation*, 251:132–142, 2015.

Hela Masri, Saoussen Krichen et Adel Guitouni : Metaheuristics for solving the biobjective single-path multicommodity communication flow problem. *International Transactions in Operational Research*, 26(2):589–614, 2019.

Siamak Moradi, Andrea Raith et Matthias Ehrgott : A bi-objective column generation algorithm for the multi-commodity minimum cost flow problem. *European Journal of Operational Research*, 244(2):369–378, 2015.

Philip James Neame : *Nonsmooth Dual Methods in Integer Programming*. University of Melbourne, Department of Mathematics and Statistics, 2000.

Lewis Ntaimo : Fenchel decomposition for stochastic mixed-integer programming. *Journal of Global Optimization*, 55(1):141–163, 2013.

Kyungchul Park, Seokhoon Kang et Sungsoo Park : An integer programming approach to the bandwidth packing problem. *Management science*, 42(9):1277–1291, 1996.

Sungsoo Park, Deokseong Kim et Kyungsik Lee : An integer programming approach to the path selection problems. *In Proceedings of the International Network Optimization Conference INOC, Evry-Paris, France*, pages 448–453, 2003.

Mark Parker et Jennifer Ryan : A column generation algorithm for bandwidth packing. *Telecommunication Systems*, 2(1):185–195, 1993.

Chao Peng, Yasuo Tan et Laurence T Yang : New algorithms for the minimum-cost single-source unsplittable flow problem. *In 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, volume 1, pages 136–141. IEEE, 2007.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa et Francois Vanderbeck : In-out separation and column generation stabilization by dual price smoothing. *In International Symposium on Experimental Algorithms*, pages 354–365. Springer, 2013.

David Pisinger : A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

Prabhakar Raghavan et Clark D Tompson : Randomized rounding : a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

G Retvdri, József J Bíró et Tibor Cinkler : A novel lagrangian-relaxation to the minimum cost multicommodity flow problem and its application to ospf traffic engineering. *In Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No. 04TH8769)*, volume 2, pages 957–962. IEEE, 2004.

Dorabella Santos, Amaro De Sousa et Filipe Alvelos : A hybrid column generation with grasp and path relinking for the network load balancing problem. *Computers & Operations Research*, 40(12):3147–3158, 2013a.

Dorabella Santos, Amaro de Sousa, Filipe Alvelos et Michal Pioro : Link load balancing optimization of telecommunication networks : A column generation based heuristic approach. *In 2010 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6. IEEE, 2010.

Dorabella Santos, Amaro de Sousa, Filipe Alvelos et Michał Pióro : Optimizing network load balancing : an hybridization approach of metaheuristics with column generation. *Telecommunication Systems*, 52(2):959–968, 2013b.

Farhad Shahrokhi et David W Matula : The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

F Bruce Shepherd et Adrian Vetta : The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems. *arXiv preprint arXiv :1504.00627*, 2015.

Martin Skutella : Approximating the single source unsplittable min-cost flow problem. *Mathematical Programming*, 91(3):493–514, 2002.

Foteini Stavropoulou, Panagiotis P Repoussis et Christos D Tarantilis : The vehicle routing problem with profits and consistency constraints. *European Journal of Operational Research*, 274(1):340–356, 2019.

Jérôme Truffot et Christophe Duhamel : A branch and price algorithm for the k-splittable maximum flow problem. *Discrete Optimization*, 5(3):629–646, 2008.

Jérôme Truffot, Christophe Duhamel et Philippe Mahey : Using branch-and-price to solve multicommodity k-splittable flow problems. *In Proceedings of International Network Optimization Conference (INOC)*. Citeseer, 2005.

Jérôme Truffot, Christophe Duhamel et Philippe Mahey : k-splittable delay constrained routing problem : A branch-and-price approach. *Networks*, 55(1):33–45, 2010.

Igor Vasilyev, Maurizio Boccia et Saïd Hanafi : An implementation of exact knapsack separation. *Journal of Global Optimization*, 66(1):127–150, 2016.

Yufei Wang et Zheng Wang : Explicit routing algorithms for internet traffic engineering. *In Proceedings Eight International Conference on Computer Communications and Networks (Cat. No. 99EX370)*, pages 582–588. IEEE, 1999.

Paul Wentges : Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

Laurence A Wolsey et George L Nemhauser : *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

Jiefeng Xu, Steve Y Chiu et Fred Glover : Tabu search for dynamic routing communications network design. *Telecommunication Systems*, 8(1):55–77, 1997.

Masoud Yaghini et Kazemzadeh MR Akhavan : A simulated annealing algorithm for unsplittable capacitated network design. 2012.

Jin Y Yen : Finding the k-shortest loopless paths in a network. *Management Science*, 17 (11):712–716, 1971.

Neal E Young : Randomized rounding without solving the linear program. *In SODA*, volume 95, pages 170–178, 1995.

Chenxia Zhao et Xianyue Li : Approximation algorithms on 0–1 linear knapsack problem with a single continuous variable. *Journal of Combinatorial optimization*, 28(4):910–916, 2014.