N° d'ordre : D.U. 1586
EDSPIC : 322

Université Blaise Pascal - Clermont-Ferrand II

École Doctorale
Sciences Pour l'Ingénieur de Clermont-Ferrand

THÈSE

présentée par

Loïc Yon

pour obtenir le grade de

Docteur d'Université

Spécialité : Informatique

Modèles et outils pour la conception stratégique de réseaux de transports publics

---

Soutenue publiquement le 23 septembre 2005 devant le jury :

M. Philippe MAHEY	Président	Professeur à l'Université Blaise Pascal
M. Aziz MOUKRIM	Rapporteur	Professeur à l'Université de Technologie de Compiègne
M. Anass NAGIH	Rapporteur	Maître de Conférences (HDR) à l'Université de Paris 13
M. Christophe DUHAMEL	Examinateur	Maître de Conférences à l'Université Blaise Pascal
M. Alain QUILLIOT	Directeur de thèse	Professeur à l'Université Blaise Pascal

À trois femmes
Blanche
Brigitte
Blondinette (Marion)

Remerciements

Je tiens tout d'abord à remercier Philippe Mahey pour avoir accepté d'être président du jury. Je remercierai ensuite Anass Nagih et Aziz Moukrim pour le travail effectué en tant que rapporteurs et pour toutes les remarques pertinentes qui m'ont permis d'améliorer le manuscrit. Christophe, tu es passé du côté obscur le jour de la soutenance en tant qu'examinateur, mais reçois mes remerciements pour toutes ces années passées. Je remercie enfin Alain Quilliot pour m'avoir permis de faire ma thèse sous sa direction.

Je remercie le "gang" des thésards (et docteurs) : Bruno, Christophe Duhamel, Christophe de Vaulx, Fabrice, Antoine et le petit dernier, bébé Léon, aussi bien dans les moments de labeurs que dans les moments moins productifs ; toutes les personnes de l'ISIMA et du LIMOS qui ont facilité la thèse et plus particulièrement Christine, Benny, Jacques, Martine ; mes compagnons ZZ : Julien, David, Tof et Omer ; Simone, Tine, Muriel, Mickaël, Ludo, Jo, François, Martin, Corinne qui m'ont permis de m'oxygéner au travers d'activités diverses comme l'escalade, le trampoline ou la danse, et de me détendre.

Une pensée va à ma famille : Jean-Pierre, Brigitte, René, Blanchette, Juliette, Gaëtan, Léonie, Marylène, Côme et Paul, la nouvelle génération. Tous s'y sont mis pour que ma thèse se termine le plus vite possible. Je remercie aussi la famille de Marion que j'ai rencontrée dans les dernières années de thèse, années où j'ai été particulièrement difficile à côtoyer. À propos, Marion, ma chère petite blondinette, non, tu ne seras pas citée dans cette section ;-) mais je vais ajouter Atchoum qui est en train de ronronner sur mes genoux.

Résumé

L'objectif de cette thèse est de proposer des méthodes et des outils pour l'aide à la conception stratégique de réseaux de transports publics en milieu urbain. Cette problématique, proche des problèmes fondateurs de la Recherche Opérationnelle, est toujours d'actualité et bénéficie du dynamisme engendré par le domaine des Télécommunications, notamment à cause de l'ouverture à la concurrence des différents marchés. Cette classe de problèmes peut être inscrite dans un cadre plus large : les problèmes de synthèse de réseaux de mobilité.

Dans un premier temps, nous proposons un état de l'art des problèmes de synthèse de réseaux ainsi qu'un panorama des différentes méthodes et outils classiquement utilisés pour la résolution de tels problèmes. Cela nous permet de formaliser la notion de problèmes de synthèse de réseaux de mobilité avec demande élastique, c'est-à-dire où les demandes d'accès au réseau varient en fonction de la qualité de service offerte par le réseau. Nous présentons différentes modélisations ainsi que des extensions possibles qui tentent de prendre en compte des critères plus réalistes : identification d'une ligne unique, prise en compte de temps d'attente ou de service, des modes de déplacement alternatifs... Ces modèles sont étudiés sur des instances de taille modeste à l'aide d'un solveur de programmes linéaires entiers mixtes. Nous proposons ensuite la résolution d'un de ces problèmes par des métaheuristiques classiques - GRASP et Tabou - afin d'obtenir de bonnes solutions sur des instances plus grandes. Dans cette optique, nous formalisons une description particulière d'un circuit baptisée "géodésique" où le circuit est décrit par la donnée d'un nombre restreint de points du circuit et d'un plus court chemin entre deux points consécutifs. La complexité du problème étudié est telle que nous avons également défini une approximation de la fonction objectif afin d'en accélérer significativement son évaluation.

La dernière partie de la thèse est consacrée à la mise en œuvre d'une méthode fondée sur le schéma de Benders pour la résolution de problèmes de synthèse de réseaux de mobilité. Cette approche fait intervenir la résolution d'un problème auxiliaire difficile, pour lequel nous proposons différentes heuristiques fondées sur la recherche de cycle augmentant. Bien que l'approche globale semble prometteuse, il reste nécessaire d'améliorer la robustesse et la précision de la résolution du problème auxiliaire.

En annexe, nous proposons une réflexion sur certaines difficultés courantes de développement logiciel pour la Recherche Opérationnelle, et nous formalisons des schémas de conception reposant sur la programmation générique qui visent à concevoir des composants logiciels à la fois flexibles et performants. Nous présentons aussi une méthode de couplage entre une méthode d'optimisation et la simulation, baptisée "couplage par enrichissement", qui diffère du couplage classique - l'optimisation de simulation - et permet d'intégrer plus de réalisme dans les modèles de synthèse de réseaux de mobilité.

Mots-clés : Transports publics urbains, synthèse de réseaux, demande élastique, programmes linéaires entiers mixtes, métaheuristiques, GRASP, Tabou, Benders, Simulation

Abstract

The aim of this thesis is to propose methods and tools to help strategic network designing dedicated to public transportation in urban environment. These issues, close to fundamental problems in Operations Research, are still topical questions and benefit from the dynamism of Telecomunications activity, especially due to the opening of the market to competition. This class of problems can fit into a larger framework: mobility network design problems.

First, we propose to survey network design problems and the existing methods and tools classically used to solve such problems. This study allows us to formalize the concept of mobility network design problems with elastic demand, that is, where the demands to access the network vary according to the quality of service provided by the network. We present various modeling as well as possible extensions that attempt to take into account more realistic criteria: identifying a single line, considering waiting times or service delays, other travelling patterns, etc. These models are studied on small-size instances with a mixed integer linear programming solver.

Then, we propose to solve one of these problems with classical metaheuristics - GRASP and Tabu - to obtain good solutions on bigger instances. From this perspective, we formalize a particular route description, named ”geodesic”, where the route is defined by a limited number of points on the route and a shortest path between two consecutive points. The complexity of the studied problem is such that we also defined an approximation of the objective function in order to significantly accelerate its evaluation.

The last part of the thesis is dedicated to the implementation of a method based on Benders' scheme to solve mobility network design problems. This approach requires the resolution of a difficult auxiliary problem, for which we propose heuristics based on the search of an increasing cycle. Although the global approach seems promising, it remains necessary to improve the robustness and the accuracy of the auxiliary problem resolution.

In the appendices, we propose considerations on some recurring issues in software design for Operations Research, and we formalize design patterns based on generic programming that aim to design both flexible and efficient software components. We also present a coupling method between optimization and simulation, named ”model enhancement”, that differs from the classical coupling - simulation optimization - and allows to integrate more realism in the models for mobility network design.

Keywords: Urban public transportation, network design, elastic demand, mixed integer programs, GRASP, Tabu, Benders, simulation

Table des matières

Liste des Algorithmes

Liste des Programmes Linéaires

Table des figures

Introduction

Le présent travail, de nature prospective, porte sur les problèmes de synthèse de réseaux qui sont susceptibles de dériver de questions concernant l'optimisation stratégique d'une offre en transports publics.

Les problèmes de synthèse de réseaux sont des problèmes d'optimisation combinatoire, qui mettent classiquement en jeu la recherche d'un objet maître "infrastructure" F qui va supporter le routage d'une famille $f = (f_i)_{i \in I}$ d'objets auxiliaires "flux", tout en respectant un certain nombre de contraintes additionnelles (comme la fiabilité, ...). Typiquement, un tel problème, que nous appellerons (*), peut le plus souvent se synthétiser très généralement comme suit :

Trouver un objet infrastructure F et un objet $f = (f_i)_{i \in I}$ famille de flux tels que :

- chaque f_i achemine une demande de flux $D_i(F)$ depuis une zone origine o_i vers une zone destination d_i dans un réseau support ;
- l'objet F supporte l'ensemble des flux $f_i, i \in I$;
- l'objet F satisfait des contraintes de structure $C(F)$;
- l'objet f satisfait des contraintes de structure $K(f)$;
- F et f minimisent un coût général $c(F, f) = c_0(F) + c_1(F, f) + c_2(F, f)$ où c_o peut être vu comme un coût d'infrastructure (amortissement), c_1 comme un coût de fonctionnement, c_2 vu comme un coût de Qualité de Service (QoS)

Pl. 1 : Problème de synthèse de réseaux

La prise en compte additionnelle d'un vecteur prix $p = (p_i)_{i \in I}$ induit ce que l'on nomme alors un problème de "tarification des réseaux". La définition du vecteur flux f_i peut porter sur des flux stationnaires agrégés, sur des flux aléatoires ou sur des flux temporisés : on s'intéresse alors aux dates de passage de chaque composante des f_i sur les différents nœuds du réseau.

L'émergence de la problématique "synthèse des réseaux" tout au long des vingt dernières années a été essentiellement le fruit des évolutions qu'a connues le monde des Télécommunications. La mise en concurrence des grands opérateurs d'infrastructures de Télécommunications

a en effet conduit ceux-ci à se poser la question du dimensionnement optimal de leur offre de transport de signaux et d'informations et de la tarification associée. Les modèles mis en œuvre ont permis, en ce qui concerne l'**objet principal** d'infrastructure F :

- la localisation des principaux dispositifs de commutation du réseau central communément appelé *backbone*,
- le dimensionnement des supports de transmission (fibres optiques, canaux hertziens, lignes RNIS, ...) de ce réseau central et leur articulation avec les boucles locales,
- le dimensionnement des réseaux virtuels associés aux différentes classes de services et de protocoles.

L'**objet auxiliaire** f exprime alors le routage des différents types de messages et de communication et leur affectation sur les différents réseaux virtuels. Les coûts c_2 (QoS) correspondent alors typiquement à des taux de perte de messages, des délais dûs à la congestion, des taux d'interruption ou de refus de transmission. Les contraintes de structure $C(F)$ traduisent, quant à elles, des préoccupations relatives à la sécurisation et à la fiabilisation des communications : possibilité de dédoublement d'une connexion sensible (solution préventive) ou de redéploiement rapide des différents flux en cas de panne (solution curative).

Quoiqu'ayant suscité des recherche théoriques visant à la recherche de solutions exactes pour les modèles induits, ces derniers ont gardé un caractère assez prospectif : ils correspondent en effet à des problèmes de décision posés à relativement long terme et s'appuient sur des données (mesure de coût, prévisions des demandes) forcément difficiles à anticiper. Leur rôle est dès lors avant tout de générer des scenarii de développement et de contribuer, à ce titre, à la chaîne de décision.

Si le dynamisme actuel de la problématique "synthèse de réseaux" s'applique en premier lieu au contexte des Télécommunications, on peut bien sûr remarquer que cette problématique peut être transplantée sur tout contexte décisionnel mettant en jeu une infrastructure, physique ou virtuelle, destinée à supporter des flux de biens, des personnes ou d'informations. On pense dès lors, naturellement, en termes d'applications, aux domaines des transports et des systèmes de production. Si l'on adopte ce point de vue, on peut d'ailleurs remarquer que les problèmes d'optimisation de tournées (voyageur de commerce, postier chinois, tournées de véhicules, ...) fondateurs de l'optimisation combinatoire, peuvent être vus comme des problèmes de synthèse de réseaux particuliers, au sein desquels l'objet auxiliaire f n'apparaît pas. L'objet de cette étude, inspirée de collaborations et d'échanges que le laboratoire a développés par le passé avec le monde des transports publics (SNCF, AIR FRANCE, SMTC, INRETS, CERTU), est d'étudier plus en détail les caractéristiques des modèles qui pourraient être utilisés par un opérateur de transports publics ayant à satisfaire des demandes disséminées dans le but d'obtenir des scenarii de réorganisation ou d'optimisation de son offre de transport.

Vers un outil d'aide à la décision ?

Ce point de vue est bien évidemment prospectif. Si la principale motivation des opérateurs de Télécommunications par rapport à une problématique de type "synthèse de réseaux" était principalement liée à la dérégulation (ouverture à la concurrence) de ce secteur, ce point de vue n'existe, en ce qui concerne les transports, qu'au niveau des opérateurs privés de fret et des opérateurs de transport aérien. De plus, le caractère social de l'activité relative au transport public dans les agglomérations rend plus délicate la mise en œuvre d'une réflexion sur

l'optimisation de celle-ci : la notion de profit, qui motive l'utilisation d'outils d'optimisation et d'aide à la décision, disparaît dans un milieu très subventionné. Pourtant, il apparaît comme probable que des marges importantes en termes de coût de fonctionnement et de qualité de service pourraient en découler. Il apparaît donc très délicat de faire accepter un outil d'aide à la décision, avec une composante CAO, dans ce domaine, tant du point au niveau des décisionnaires qui n'en voient pas forcément l'usage, que du point de vue du public dont les besoins sont très difficiles à estimer (notamment à cause de la difficulté d'acquisition des données). Cependant, la donne politique est peut-être en train de changer et on a vu récemment se mettre en place des méthodes d'optimisation pour réduire les coûts de fonctionnement de services publics bien particuliers, dédiés aux personnes à mobilité réduite, le transport à la demande.

Intégrer une composante CAO au niveau de la décision stratégique concernant l'offre de transport d'un opérateur, suppose d'être capable de modéliser efficacement l'impact d'une modification de cette offre sur les demandes et sur les coûts. Cette dernière problématique de nature prévisionnelle est en soi très difficile à traiter dans le cas des transports dans la mesure où la demande du public par rapport à un mode donné est assez élastique, à la fois aux prix et aux conditions techniques du transport. Cette difficulté se trouve accrue par le fait que, contrairement aux opérateurs de Télécommunications, les opérateurs de transports publics n'ont (à l'exception des opérateurs aériens) pas structuré leur système d'information de façon à être en condition pour acquérir, stocker et exploiter de façon systématique les données utiles pour la modélisation des coûts et des demandes. Et même dans le cas contraire, s'il était possible de connaître les besoins des usagers, il est quasiment impossible de connaître ceux des personnes qui n'utilisent pas le système de transports mais qui seraient susceptibles de l'emprunter si le système leur semblait utile.

Si l'on met ces interrogations de côté, et que l'on se place donc dans une perspective qui voit s'affirmer la pertinence d'une aide CAO à la conception de l'offre stratégique en transports publics de la part d'un opérateur donné, alors on voit en se référant au cadre générique (*) que :

- l'objet F doit fournir la description de la partie flexible de l'offre de transport proposée par l'opérateur. Il se définira donc, à partir d'un réseau d'infrastructures donné, (rails, routes, couloirs aériens, ...) comme un ensemble de "lignes" (c'est-à-dire des circuits parcourus par des véhicules de même type). Ces véhicules seront perçus comme étant un même véhicule par l'usager. Suivant le type de problème, l'articulation avec le temps pourra se faire de différentes manières. Au niveau urbain, la notion de temps apparaîtra plutôt comme des fréquences et de vitesses de déplacement. La densité du tissu urbain permet en effet de considérer que les véhicules évoluent de façon stationnaire. Dès que l'espace géographique s'étend - zone péri-urbaines, régionales, nationales ou internationales, se poseront des problèmes de connexions rapides et de synchronisation entre les différentes lignes et on fera donc intervenir des dates de passage et/ou des fenêtres de temps.
- la demande usager, qui pourra être discriminée selon les catégories d'usagers, devra être corrélée d'une façon ou d'une autre aux temps de parcours et aux horaires. Ceci pourra se faire soit par insertion de contraintes relatives aux fenêtres de temps associées aux parcours, soit par l'introduction de mesure d'élasticité des demandes aux temps de parcours. L'objet f représentera dès lors les flux usagers discriminés selon les couples origine-destination, les horaires et les caractéristiques des usagers.

- la contrainte de couverture qui relie l'infrastructure F à l'objet représentant le routage induit f sera partielle, c'est-à-dire qu'elle ne concernera que certaines portions de mouvement global du routage. Ceci correspond au fait qu'un déplacement usager s'effectue en général de façon multimodale, certains modes utilisés étant propres à l'usager (véhicule personnel, marche à pied, etc).
- les coûts de fonctionnement seront en soi difficiles à modéliser. En effet, si les coûts fixes, de maintenance ou d'amortissement sont plus facilement quantifiables, certains coûts, incompressibles, ne le sont pas : on peut citer par exemple, lors d'une modification d'une offre de transport, tous les coûts liés à la masse salariale, aussi bien les conducteurs des véhicules que les agents de support du service (agents de maintenance, de chargement, commerciaux, hôtesses, contrôleurs ...)
- les coûts de qualité de service (QoS) seront essentiellement définis par les temps de parcours des usagers, par le nombre de changements de véhicules.

Cet ensemble de caractéristiques va définir une classe de problèmes de synthèse de réseaux dans lesquels les temps et horaires de parcours induits vont constituer la principale évaluation "usager", tandis que, du point de vue de l'opérateur, c'est la capacité du système à mutualiser de façon forte leurs parcours qui définira la qualité de ce système. En ce qui concerne les modèles et les algorithmes, on trouvera là les principales différences par rapport aux modèles et algorithmes associés à l'univers des Télécommunications, et ainsi en récapitulant :

- les contraintes structurelles sur l'objet F tendent à être des contraintes de flot,
- la couverture de f par F pourra être partielle, ce qui signifiera qu'à la limite, l'offre pourra même être inexistante,
- le caractère "non fractionnable" des véhicules fait que le principal critère de qualité d'une solution résidera dans sa capacité à induire des acheminements usagers rapides au moyen des véhicules bien remplis.

Le but de notre étude va donc être de proposer un certains nombre de modèles visant à traiter cette classe de problèmes et de les comparer entre eux, tant du point de vue des résultats qu'ils induisent, que de leur sensibilité, et des techniques algorithmiques que l'on peut mettre en œuvre pour les traiter. Les modèles se différencient suivant :

- la représentation de l'objet maître circulant F : entier/fractionnaire, flot entier/ flot fractionnaire, famille de circuits accompagnés de fréquences, famille de circuits temporisés (accompagnés de temps de passage aux nœuds) et son articulation à l'espace temps. Ces différentes représentations diffèrent en ce sens qu'elles s'attachent plus ou moins au véhicule et à son parcours, pouvant, dans un cas extrême, ramener la description du système de transport à une famille d'arcs du réseau munis de coefficients de vitesse et de capacité,
- la représentation de la demande des usagers et de sa dépendance au temps. Ces demandes peuvent être plus ou moins éclatées (plus ou moins grand nombre de couples origine-destination) et en relation avec le temps par le biais de fonctions de demandes élastiques relatives au temps, de demandes fixes couplées à des bornes supérieures sur le temps d'acheminement, de fonctions de performance exprimant l'intérêt qu'a le système d'assurer une connexion rapide entre certains couples origine-destination,
- les mesures de coûts économiques (c_0 et c_1 dans le cadre générique(*)), pourront être linéaires en F et f, concaves ou encore de façon à restituer les caractéristiques économiques d'un système assez complexe, linéaire par morceaux ou en escalier.

Comparer les résultats associés à deux modèles distincts est pertinent dans la mesure où, pour un même problème pratique, la complexité élevée du problème fait que l'on peut envisager de le traiter par le biais de modèles très sensiblement différents, et qu'il est dès lors, intéressant de pouvoir prévoir en quoi ces modèles induisent des solutions relativement semblables ou, au contraire, très différentes.

Plan de l'étude

Au chapitre 1, nous décrirons la problématique de synthèse de réseaux en détail ainsi que des méthodes couramment employées pour la résolution de tels problèmes. Dans la mesure où nous nous intéressons particulièrement à la synthèse de réseaux de mobilité, nous ferons aussi un état de l'art des différents problèmes de Tournées de Véhicules.

Le chapitre 2 est consacré à la formalisation du problème de transport que constitue la synthèse de réseaux de mobilité. Nous donnerons un modèle général lorsque la demande est élastique dont on tirera deux spécialisations. Notre propos sera illustré par des expérimentations numériques avec la résolution exacte de ces modèles avec un solveur de programmes entiers mixtes.

Cette méthode de résolution atteint très vite ses limites. Ainsi, on étudiera au chapitre 3, les adaptations de deux métaheuristiques désormais classiques, la méthode GRASP et la méthode tabou, au problème de synthèse de réseaux de mobilité. L'originalité de notre implémentation réside dans la structure de tournée employée ; nous avons baptisée celle-ci géodésique.

Au chapitre 5, nous présenterons et analyserons un modèle particulier de flot et multiflot couplés que nous appliquerons à des problèmes dotés de contraintes temporelles (réseaux dynamiques) et pour le traitement duquel nous introduirons une technique d'agrégation. Nous conclurons par un bilan de l'étude et un état des perspectives associées à cette problématique des modèles d'optimisation de réseaux appliqués à la conception de systèmes de transports publics.

Outres quelques rappels sur les notions fondamentales et des commentaires sur l'implémentation et sur les tests, le lecteur trouvera en annexes deux autres points développés au cours de cette thèse : le premier est la description et la validation d'un mécanisme d'interaction entre la simulation par événements discrets et les techniques de résolution en Recherche Opérationnelle au travers du cas pratique que constitue le problème de synthèse de réseaux de mobilité. Le second point est un essai de formalisation de concepts de programmation pour disposer d'algorithmes génériques, exploitables dans de nombreuses conditions, et qui soient aussi efficaces.

Chapitre 1

Synthèse de réseau : application aux réseaux de mobilité

Sommaire

Ce chapitre est consacré à l'état de l'art sur un problème particulier d'optimisation dans les réseaux : la **synthèse de réseaux**. Nous verrons tout d'abord ce que l'on appelle la synthèse de réseaux et les **méthodes utilisées** pour la résolution de ce type de problème. Bien entendu, on ne pourra être exhaustif tant le domaine est large et on pourra se référer par exemple à [QUI05] pour un approfondissement. Nous nous intéresserons ensuite à un problème de synthèse de réseaux sur des réseaux particuliers : les **réseaux de mobilité**. Après une rapide définition, nous ferons un rapide survol d'un problème connexe, le problème de **Tournées de Véhicules**, ainsi que ses différentes extensions.

1.1. Synthèse de réseaux

Le problème de synthèse de réseaux n'est qu'un aspect des différents problèmes qui se posent dans l'optimisation dans un réseau. En effet, d'après [GLON04], on peut distinguer quatre grands types de problèmes d'optimisation sur les réseaux : les problèmes de synthèse de réseaux (*Network design*), les problèmes de dimensionnement de réseaux (*Network loading*), les problèmes de conception de réseaux et les problèmes d'allocation de ressources.

On appelle problème de **synthèse de réseaux**, tout problème qui consiste à identifier un sous-graphe optimal d'un graphe relativement à des contraintes, i.e. de déterminer la topologie du réseau (identifier les nœuds ou les arcs dont le rôle est clef) le plus généralement au coût le plus faible. Bien souvent, on ne cherche pas à modéliser l'écoulement de trafic et ces modèles sont sans capacité. Ainsi, on peut voir les problèmes "classiques" tels que la recherche d'un arbre couvrant (problème facile, soluble en temps polynomial), d'un stable, d'un cycle hamiltonien (problème NP-difficile) ou bien encore des plus courts chemins sont des exemples très connus de cette classe de problème.

Dans les problèmes de **dimensionnement de réseaux**, on suppose que la topologie est donnée et qu'il faut dimensionner, c'est-à-dire installer des capacités de manière à permettre l'écoulement d'un trafic connu à l'avance (voir, par exemple, [BCGT98]). Les problèmes de **conception de réseaux** sont des problèmes où il faut à la fois déterminer la topologie et dimensionner. Il faut alors parfois appliquer des contraintes supplémentaires (politique de routage spéciale, topologie déterminée, exigence relative à la robustesse ou à la sécurité,...). Les problèmes d'**allocation de ressources** supposent que le réseau soit construit correctement et dimensionné. On cherche à allouer des capacités aux différents services qui demandent à utiliser le réseau.

Tous ces problèmes sont applicables, par exemple, en Télécommunications [GLON04] et choisir tel ou tel problème dépend de l'horizon de temps auquel il s'applique : du long terme (années) pour la synthèse au court terme (quelques semaines ou quelques mois) pour l'allocation. Mais on les retrouve aussi, bien entendu, dans les domaines des transports, de la productique, de la conception de circuits électroniques ou des réseaux informatiques [FG94].

Si les problèmes ne concernent pas simplement les nœuds du graphe mais des sous-ensembles de nœuds, on parle alors de **problèmes** [de synthèse, de conception, ...] **généralisés** [CL03].

1.1.1. Problème CFA

De nombreux problèmes en transports ou en télécommunications peuvent s'écrire sous la forme d'un problème d'affection de flot et de capacités, ou encore *Capacity and Flow Assignment Problem (CFA)* dès que l'on a identifié des nœuds ou des arcs supports de canaux de communication. Gerla est le premier à avoir formulé le problème dans sa thèse [GER73].

Trouver sur un réseau $G = (V, E)$ un flot infrastructure $F \geq 0$ et un multiflot $f = (f^k)^{k \in K} \geq$ tels que :

- f respecte des contraintes structurelles/topologiques ;
- chaque composante f^k du multiflot f achemine une certaine quantité de flot d'un ensemble d'origines vers un ensemble de destinations ;
- f et F respectent une relation de couplage pour tout arc $e \in E$;
- une certaine fonction coût soit minimale.

Pl. 2 : Problème CFA

Les contraintes structurelles sur l'objet infrastructure peuvent être de différentes natures. On peut, par exemple, chercher à assurer une certaine sécurité sur le réseau et que tout produit circulant sur le réseau dispose d'au moins deux chemins potentiels arcs disjoints.

La relation de couplage entre le flot F et le multiflot f peut être construite avec différents opérateurs : l'agrégation ($\sum_k f_e^k \leq F_e, \forall e$), le sup ($f_e^k \leq F_e, \forall k \forall e$), ...

En général, le coût prend en compte différents paramètres : le **coût d'installation** de l'infrastructure F, des **coûts d'utilisation** de F par f et des coûts liés au **défaut de service**. Les coûts sont le plus souvent concaves et dégressifs pour traduire l'**économie d'échelle**. Les coûts de défaut de service essayent de synthétiser les taux de pertes ou de délais induits par un encombrement excessif du réseau (**congestion**). La fonction de Kleinrock [KLE75] est la fonction la plus connue (même si elle n'est pas toujours la plus réaliste). Mahey *et al.* [MBB01] résolvent le problème en utilisant la décomposition de Benders.

1.1.2. De nombreuses facettes

Le réseau sous-jacent pourra être **dynamique** ou **temporisé**, c'est-à-dire intégrer une composante temps en proposant, par exemple, une indexation des nœuds ou des arcs suivant l'instant considéré (on considère une copie d'un sommet ou d'un arc pour chaque instant discrétisé).

Pour traiter l'acheminement de produits hétérogènes, ou traduire la multimodalité d'un réseau (différents moyens de communication), on pourra considérer des réseaux **multicouches** [BMM94].

Si l'infrastructure F est supposée connue et fixée, cela revient à ne considérer que la problématique du **routage** des produits f.

Si, au contraire, la topologie support de F n'est pas complètement déterminée, il faut introduire un problème de **localisation** (*location problem*). Cette classe de problèmes est utilisée pour initialiser certaines étapes de problèmes où la notion de couverture (en infrastructures ou en ressources) est importante (notamment lorsque l'on s'intéresse aux services publics [BLS03]).

On peut aussi imposer des contraintes sur la **capacité** des arcs [GEN95, GCF98] et définir ainsi des problèmes d'**expansion** [GC95].

Enfin, si l'on s'intéresse plutôt à la problématique de **tarification**, il faut introduire un vecteur prix applicable sur le routage f.

Comme on peut donc le voir, la famille de problèmes de synthèse de réseaux est très large et elle a été particulièrement étudiée ces dernières années avec l'essor des Télécommunications. Mais ces problèmes restent difficiles à résoudre pour de trop nombreuses raisons. Le plus souvent :

- ils sont de très **grande taille**,
- ils sont **mal conditionnés** et induisent des phénomènes de dégénérescence,
- ils admettent plusieurs optima locaux,
- les contraintes, notamment structurelles, peuvent être très complexes à gérer [PD98],
- l'objet infrastructure est **entier**.

Ces caractéristiques soulignent aussi l'intérêt des différentes **méthodes de décomposition** (structure hiérarchique, hétérogénéité du réseau ou des produits, ...) et des méthodes de **synthèse de l'information** sur le routage (travailler avec un flot agrégé par exemple, [BMQ03]).

1.2. Méthodes

Dans cette section, nous allons voir les différentes méthodes de résolution des problèmes de synthèse de réseaux. Nous les avons classées en différentes familles : les méthodes issues de la programmation linéaire et les approches polyédrales, les formulations flot et multiflot, les réseaux de file d'attente, les modèles de théorie des jeux et la simulation (même si cette dernière n'est pas vraiment une technique de résolution).

1.2.1. Programmation linéaire et polyèdres

Les problèmes de synthèse de réseaux se modélisent souvent comme des programmes entiers "mixtes" où interviennent des variables discrètes et des variables continues. La résolution de ces problèmes mène à une utilisation intensive des outils de l'optimisation continue et de la programmation linéaire.

Décomposition de Benders

On rappelle succinctement le principe de la décomposition de **Benders** [BEN62, GEO72] dans la mesure où cette méthode sera utilisée au chapitre 4. C'est une méthode qui consiste à **isoler** dans le problème à résoudre, par exemple le PL 3, les **différentes variables suivant leur type** : classiquement, les variables entières d'un côté et les variables réelles de l'autre. Les variables entières sont fixées dans un problème auxiliaire qui doit être facile à résoudre et on rajoute des contraintes pour l'exploration de celles-ci.

Considérons le programme linéaire à résoudre suivant :

Trouver un vecteur réel x et un vecteur entier y tels que $c.x + f.y$ soit **minimale** sous les contraintes

$$Ax + By \geq b \tag{3.1}$$
$$y \in Y \tag{3.2}$$
$$x \geq 0 \tag{3.3}$$

où c et f sont des vecteurs coût, A, B des matrices, b un vecteur et Y un sous-ensemble de $\mathbb{N}$

Pl. 3 : Benders, problème entier mixte à résoudre

Si on fixe la variable y (réalisable pour le problème), on doit alors résoudre le problème restreint (Pl 4). On peut aussi tout à fait résoudre le dual du programme restreint (Pl 5).

Trouver x qui **minimise** $c.x$ sous les contraintes

$$Ax \geq b - By \tag{4.1}$$
$$x \geq 0 \tag{4.2}$$

Pl. 4 : Benders, problème restreint

Trouver u qui **maximise** $(b - By).u$ sous les contraintes

$$A^T u \leq c \tag{5.1}$$
$$u \geq 0 \tag{5.2}$$

Pl. 5 : Benders, dual du problème restreint

L'avantage de résoudre le dual — et non le primal — est le fait que la variable fixée y n'intervient que dans la fonction objectif. On peut donc réécrire le problème d'optimisation initial comme :

$$\min_{y \in Y} \left[f.y + \min_{x \geq 0} \{c.x | Ax \geq b - By\} \right] \textbf{ ou encore } \min_{y \in Y} \left[f.y + \max_{u \geq 0} \left\{ (b - By).u | A^T u \geq c \right\} \right]$$

Le principe de la méthode de résolution de Benders est donc d'itérer la résolution successive de deux problèmes : le problème maître (Pl 6) et le problème esclave (Pl 7). Le problème maître est alimenté en contraintes (6.1) et (6.2) à chaque résolution du problème esclave. Si le problème esclave est borné, une solution optimale $\overline{u}_k$ permet de rajouter la contrainte d'**amélioration** (6.1). Si, au contraire, le problème esclave n'est pas borné, la direction non bornée $\tilde{u}_\ell$ permet d'ajouter la contrainte de **réalisabilité** (6.2).

min z
sous les contraintes

$$z \geq f.y + (b - By).\overline{u}_k \quad k = 1..K \tag{6.1}$$
$$(b - By).\tilde{u}_\ell \leq 0 \quad \ell = 1..L \tag{6.2}$$
$$y \in Y \tag{6.3}$$

Pl. 6 : Benders, problème maître

max $f.y + (b - By).u$
sous les contraintes

$$A^T u \leq c \tag{7.1}$$
$$u \geq 0 \tag{7.2}$$

Pl. 7 : Benders, problème esclave

L'algorithme de principe de la méthode est le suivant :

Algorithme 1 : Algorithme de la décomposition de Benders

```
⇝ initialisations
y ← solution entière réalisable
BI ← −∞ // Borne Inférieure
BS ← +∞ // Borne Supérieure
tant que BS − BI > ε faire
    ⇝ Résolution du sous-problème
    Résoudre le problème dual à y fixé (Pl. 7)
    si le problème n'est pas borné alors
        Déterminer la direction ũ_ℓ du rayon extrême
        Ajouter la coupe (b − By)^T ũ_ℓ ≤ 0 au problème maître (L ← L + 1)
    sinon
        Calculer la solution optimale ū_k
        Ajouter la coupe z ≥ f.y + (b − By)^T ū_k au problème maître (K ← K + 1)
        BS ← min{BS, f.y + (b − By)^T ū}
    fin si
    ⇝ Résolution du problème principal (Pl. 6)
    min_y {z | coupes, y ∈ Y}
    BI ← z̄
fin tant que
```

La décomposition de Benders pour les problèmes entiers mixtes est particulièrement utile lorsque le sous-problème et le problème maître restreint sont faciles à résoudre et que le problème original ne l'est pas. Pour des applications à la synthèse de réseaux, on pourra se référer, par exemple, à [BOY97, BFG98, MBB01].

Décomposition Lagrangienne

La **décomposition Lagrangienne** est particulièrement utilisée lors du processus de résolution des problèmes de synthèse de réseaux, car elle permet de **séparer** le problème initial en plusieurs sous-problèmes en relâchant certaines contraintes, dites couplantes. Les contraintes relâchées sont réinjectées dans les fonctions objectifs pondérées par des coefficients : les multiplicateurs de Lagrange. La décomposition Lagrangienne fait appel à la méthode de relaxation de Lagrange expliquée au point suivant.

Illustrons la décomposition Lagrangienne sur un exemple simple :

Minimiser $z = c.x$
sous les contraintes

$$A_1 x \leq b_1 \quad (8.1)$$
$$A_2 x \leq b_2 \quad (8.2)$$
$$x \geq 0 \quad (8.3)$$

Pl. 8 : Problème d'optimisation (P) à résoudre

On introduit une contrainte (bloc) $x = y$ dans le programme (P). On obtient alors le programme (P'_1). Par suite, on dualise les contraintes $x = y$ et le problème (P'_2) obtenu peut être résolu par la relaxation Lagrangienne :

Minimiser $z = c.x$
sous les contraintes

$$A_1 x \leq b_1 \quad (9.1)$$
$$A_2 y \leq b_2 \quad (9.2)$$
$$x = y \quad (9.3)$$
$$x \geq 0 \quad (9.4)$$

Pl. 9 : Problème (P'_1)

Minimiser $z = c.x + \lambda(y - x)$
sous les contraintes

$$A_1 x \leq b_1 \quad (10.1)$$
$$A_2 y \leq b_2 \quad (10.2)$$
$$x, y \geq 0 \quad (10.3)$$

Pl. 10 : Problème (P'_2)

Le problème donné en exemple peut donc se décomposer en deux blocs indépendants : l'un en x, l'autre en y.

Relaxation Lagrangienne

La méthode de **relaxation lagrangienne** est une méthode générale de résolution de problème d'optimisation sous contraintes, et particulièrement en optimisation linéaire et combinatoire. Soient $\mathcal{X} \subset \mathbb{R}^n$, $f : \mathbb{R}^n \mapsto \mathbb{R}$ et $g : \mathbb{R}^n \mapsto \mathbb{R}^m$. Considérons le problème (P) d'optimisation sous contraintes général suivant :

Minimiser $f(x)$
sous les contraintes

$$g_i(x) \leq 0 \quad \forall i \in \{1, , m\} \quad (11.1)$$
$$x \in \mathcal{X} \quad (11.2)$$

Pl. 11 : Lagrange, problème d'optimisation sous contrainte (P)

On suppose que les contraintes (11.2) sont des contraintes "faciles", tandis que les contraintes (11.1) sont les contraintes "difficiles", sur lesquelles nous allons appliquer la relaxation lagrangienne.

On associe à nos contraintes (11.1), un vecteur de multiplicateurs $\lambda \in \mathbb{R}^{+m}$ et on considère la *fonction de Lagrange* (ou Lagrangien) suivante :

$$L(x, \lambda) = f(x) + \lambda^T g(x) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \quad \forall x \in \mathcal{X}, \lambda \in \mathbb{R}^m \tag{11.3}$$

et on considère le problème relâché suivant :

$$(P_\lambda) \quad \min_{x \in \mathcal{X}} L(x, \lambda)$$

où $\lambda_i \geq 0$ est le "prix" à payer lorsque la i-ième contrainte est violée (on parle aussi de pénalité).

Le problème relâché est, pour un vecteur de multiplicateurs donné, plus simple à résoudre car il ne reste plus que les contraintes "faciles". Par ailleurs, pour un vecteur de multiplicateurs λ donné, une solution optimale $x^*(\lambda)$ du problème relâché est une solution du problème primal si les deux conditions suivantes sont vérifiées :

$$(i)\ x^*(\lambda) \text{ est réalisable pour le problème primal } (P), \tag{11.4}$$

$$(ii)\ \lambda^T g(x) = 0 \quad \text{(écarts complémentaires)}. \tag{11.5}$$

Soit $\mathcal{L}(\lambda) = \min_{x \in \mathcal{X}} L(x, \lambda)$, la *fonction duale* de P. Pour tout vecteur de multiplicateurs $\lambda \geq 0$, la fonction duale est une borne inférieure pour le problème initial. Et pour toute solution x réalisable pour le problème (P), nous avons l'inégalité suivante :

$$\mathcal{L}(\lambda) \leq f(x^*) \leq f(x) \tag{11.6}$$

où x^* désigne une solution optimale pour le problème initial (P).

Le problème lagrangien dual consiste à trouver le vecteur des multiplicateurs qui maximise la fonction duale $\mathcal{L}$.

$$(\mathrm{D}) \quad \max_{\lambda \geq 0} \mathcal{L}(\lambda)$$

D'après l'inégalité (11.6), la valeur optimale $\mathcal{L}^*$ du problème dual est une borne inférieure pour le problème primal : **la solution optimale du dual n'est pas nécessairement réalisable pour le primal** (dualité faible). Le gap $f(x^*) - \mathcal{L}^*$ sera alors non nul. Le problème dual est communément résolu par la méthode des sous-gradients ou la méthode des faisceaux.

Génération de colonnes

Les méthodes de **génération de colonnes** sont particulièrement adaptées pour les problèmes dont la modélisation fait intervenir un très grand nombre de variables (par exemple, les modélisations par arcs-chemins) [DL02] . Le principe de la méthode est de ne pas chercher à expliciter l'ensemble des variables — encore appelées *colonnes* — mais à travailler seulement avec un petit sous-ensemble, dit variables en base. Le processus de résolution, itératif, vise à faire entrer de nouvelles colonnes en base en analysant, par exemple, les coûts réduits.

$$\text{Minimiser } c(x) = \sum_{j \in J^*} c_j x_j$$
$$\text{sous la contrainte}$$
$$\sum_{j \in J^*} A_j x_j = b \qquad (12.1)$$

PL. 12 : GC, Problème initial (P)

$$\text{Minimiser } c(x) = \sum_{j \in J} c_j x_j$$
$$\text{sous la contrainte}$$
$$\sum_{j \in J} A_j x_j = b \qquad (13.1)$$

PL. 13 : GC, Problème "réduit" (P_J)

On cherche à résoudre le problème (P). L'ensemble J^* est **très grand** et cela rend le problème difficile à résoudre. On s'intéresse donc au problème où l'on ne résout pas sur J^* mais sur un sous-ensemble $J \subseteq J^*$.

Ainsi, on résout le problème (P_J) à l'optimum et on extrait les variables duales (π). On cherche alors une colonne $j \notin J$ telle que le coût réduit associé est négatif, ou encore $c(j) - \pi A_j < 0$. Si une telle colonne existe, on l'ajoute à l'ensemble J et le nouveau problème (P_J) est résolu grâce à des opération de pivotage. Sinon, cela veut dire que l'on dispose déjà de la solution optimale du problème initial (P).

Dans le cas général, on considère que l'on dispose d'un **"oracle"** qui permet de répondre à la question de savoir s'il existe une colonne entrante ou pas, c'est-à-dire si la colonne vérifie une certaine propriété qui la rend admissible dans le problème réduit. La génération de colonnes a été introduite par Dantzig et Wolfe [DW60] d'une part et Gilmore et Gomory [GG61, GG63] d'autre part pour résoudre le problème de *Cutting Stock Problem* où le problème de colonne entrante est un problème de sac-à-dos résolu grâce à la programmation dynamique.

Décomposition de Dantzig-Wolfe

La méthode de décomposition de **Dantzig-Wolfe** [DW60, LAS70] est une méthode qui s'appuie sur le principe de la génération de colonnes. Elle consiste en premier lieu à identifier les contraintes couplantes dans le problème et à décomposer le reste des contraintes en blocs indépendants. On reformule alors chaque bloc en terme de points extrêmes, dont le nombre généralement exponentiel rend nécessaire l'usage de la génération de colonnes. Le programme maître conserve les contraintes couplantes réexprimées en terme de combinaison convexe de points extrêmes. L'idée consiste alors à résoudre le programme maître puis à rechercher un point extrême améliorant dans chacun des sous-problèmes.

On suppose que l'on doit résoudre un problème de la forme de (P) donné par le (PL 14). Les contraintes couplantes sont identifiées. Les c_i sont des vecteurs coûts. Les x_i sont des vecteurs

d'inconnues. Chacun de ces vecteurs est associé à un bloc de contraintes indépendantes. Les F_i et D_i sont des matrices données.

$$
\begin{array}{llllllllll}
max & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_tx_t & & \\
\text{sous les contraintes} & & & & & & & & & \\
 & F_1x_1 & & & & & & = & b_1 & (14.1) \\
 & & & F_2x_2 & & & & = & b_2 & (14.2) \\
 & & & & & \ddots & & \vdots & \vdots & \vdots \\
 & & & & & & F_tx_t & = & b_t & (14.t) \\
 & D_1x_1 & & D_1x_1 & & \cdots & D_1x_1 & = & b_0 & (14.0) \\
 & x_1 & , & x_2 & , & \cdots & , \quad x_t & \geq & 0 &
\end{array}
$$

Pl. 14 : Dantzig-Wolfe, un problème type (P) à résoudre

Chacun des sous-problèmes suivants doit être facilement soluble sinon le schéma de décomposition n'est pas adapté :

$$
\begin{array}{lllll}
max & c_ix_i & & & \\
sous & F_ix_i & = & b_i & \\
 & x_i & \geq & 0 &
\end{array}
$$

Le résultat est, soit un point extrême x_i^j du domaine réalisable, soit un rayon extrême w_i^k lorsque le problème n'est pas borné. Si l'on appelle P_i l'ensemble des solutions réalisables pour le sous-problème i, l'ensemble des points extrêmes J_i et K_i l'ensemble des rayons extrêmes, alors toute solution réalisable $x_i \in P_i$ peut s'écrire :

$$x_i = \sum_{j \in J_i} \lambda_i^j x_i^j + \sum_{k \in K_i} \theta_i^k w_i^k \text{ où } \theta_i^k \geq 0,\ \lambda_i^j \geq 0 \text{ et } \sum_{j \in J_i} \lambda_i^j = 1$$

Pour écrire le problème maître de la décomposition, il suffit de réinjecter l'expression des points x_i en termes de θ et de λ et d'enlever les contraintes faciles — les contraintes de blocs repérées de (14.1) à (14.t) —, il ne reste plus que les contraintes "difficiles" notées (14.0). Le problème maître a potentiellement un très grand nombre de variables, c'est pourquoi cette décomposition utilise la méthode de génération de colonnes en partant du principe que la génération de colonnes est, de toute façon, une opération moins coûteuse que la résolution du problème initial. Il faut juste calculer suffisamment de points extrêmes pour pouvoir trouver une base initiale suffisante.

Décomposition Proximale

La méthode de **décomposition proximale** [CMR94] est une méthode plus générale que la méthode de décomposition de Dantzig-Wolfe. Elle est particulièrement bien adaptée aux **problèmes de routage non linéaires** si le problème se découpe en problèmes indépendants reliés par une contrainte de couplage.

Minimiser $z = f(x)$
sous la contrainte

$$x \in A \tag{15.1}$$

où f est une fonction convexe séparable

Pl. 15 : Méthode proximale, Problème (P)

Si $A^{\perp}$ désigne le sous-espace orthogonal de A, la paire primale-duale (x^*, y^*) solution optimale du problème et de son dual doit appartenir à l'espace produit cartésien $A \times A^{\perp}$. Pour trouver cette paire primale-duale optimale, l'algorithme procède de manière itérative. Chaque itération comporte deux étapes. La première, l'étape proximale, consiste à minimiser sans la contrainte (15.1) la fonction objectif régularisée, c'est-à-dire à laquelle a été ajouté un terme quadratique appelé "terme proximal". La seconde effectue la projection des solutions de l'étape précédente sur l'espace $A \times A^{\perp}$.

L'efficacité de l'algorithme de décomposition proximale dépend étroitement de la formulation judicieuse du problème [OUO95, BOY97].

Techniques de branchement

En optimisation combinatoire (une partie de l'objet cherché est discret), les techniques de branchement, le *branch and bound* [LD60] et ses dérivées, sont très utilisées. Ces méthodes, arborescentes, visent à couvrir l'ensemble des solutions réalisables de manière plus intelligente et surtout plus efficace que la simple énumération.

Le **branchement**, ou *branching*, est une méthode pour déterminer l'ensemble des solutions réalisables, idéalement découpé en sous-régions. Il faut alors, dans la mesure du possible, être capable de donner une borne inférieure et une borne supérieure pour chaque sous-région évaluée (*bounding*). Le processus est itératif, sous forme d'arbre et chaque sous-région est découpée pour affiner la recherche (si possible en ensembles pas trop petits). Si l'on cherche à minimiser une fonction, il suffit de garder en mémoire la meilleure borne réalisable (la plus petite borne sup par exemple) et on peut éliminer avec certitude toute région dont la borne inf est plus grande que cette valeur. Le processus de résolution s'arrête lorsque tous les nœuds ont été parcourus ou éliminés.

Les méthodes dérivées ne font que changer la manière de résoudre le problème relâché et sont plus efficaces encore. Elles portent le nom de *Branch and Cut* [GJR84], *Branch and Price* [BJN+98], ou encore *Branch and Cut and Price*. Dans le *Branch and Cut*, on recherche de nouvelles inégalités ou coupes (idéalement des facettes du polyèdre solution) pour accélérer la recherche arborescente. Le *Branch & Price* fait appel à la génération de colonnes pour résoudre le problème relâché et la dernière méthode combine les précédentes : coupes et génération de colonnes.

Polyèdres

Les **méthodes polyédrales** [MAJ05] sont très utilisées pour la recherche de l'objet difficile qu'est l'objet infrastructure du problème de synthèse de réseaux.

L'idée de ces méthodes est de ramener le problème d'optimisation combinatoire en question à la résolution d'un programme linéaire, par la description de l'enveloppe convexe de ses solutions par un système d'inégalités linéaires. Chaque solution peut être représentée par un vecteur en $\{0, 1\}$. En considérant ces vecteurs comme des points de $\mathbb{R}$ et en déterminant le polyèdre, enveloppe convexe de ces points (i.e. en déterminant les contraintes linéaires qui définissent les hyperplans d'appui de ce polyèdre), on ramène le problème à l'optimisation d'une fonction linéaire sur ce polyèdre. Cette approche, qui a été véritablement introduite par les travaux d'Edmonds sur le problème du couplage, a été appliquée par la suite avec succès à plusieurs problèmes d'optimisation combinatoire comme le problème du voyageur de commerce et le problème de la coupe maximum. L'approche polyédrale peut être considérée aussi dans le cadre plus général des méthodes de coupes en programmation en nombres entiers. Les algorithmes fondés sur les techniques polyédrales sont généralement connus sous le nom d'algorithmes de coupes et de branchements (*Branch & Cut*).

Une étape cruciale dans les méthodes polyédrales concerne l'identification des contraintes qui définissent des hyperplans d'appui du polyèdre des solutions. Si le problème est NP-difficile, il y a généralement peu d'espoir d'obtenir une caractérisation complète de ces contraintes. Cependant, une description partielle de ces contraintes peut être suffisante pour résoudre le problème en temps polynomial. En effet, ces contraintes peuvent être utilisées dans une méthode de coupes pour résoudre le problème. Cette méthode nécessite uniquement un algorithme polynomial pour résoudre le problème de séparation associé aux contraintes. Ce problème consiste à vérifier si une solution donnée satisfait les contraintes ou, dans le cas contraire, de déterminer une contrainte violée. Si le problème de séparation peut être résolu en temps polynomial, alors, d'après la méthode des ellipsoïdes, le programme linéaire défini par ces contraintes (dont pourtant le nombre est souvent exponentiel) peut être lui-aussi résolu en temps polynomial.

L'idée générale de la méthode de coupes, *Branch & Cut*, est de résoudre la relaxation linéaire du problème, c'est-à-dire le programme linéaire obtenu en relâchant les contraintes d'intégrité du problème combinatoire. On peut remarquer que si sa solution optimale est entière, alors elle est également une solution optimale pour le problème en question. Si ce n'est pas le cas, on recherche une (ou plusieurs) contraintes valides pour le problème et qui soient violées par la solution optimale du problème relaxé. On ajoute alors ces contraintes au programme linéaire. Si la solution optimale du nouveau programme est entière, alors on aura résolu le problème, sinon on détermine à nouveau des contraintes violées qui seront ajoutées au programme et ainsi de suite. A la fin de ce processus, soit on obtient solution entière et donc on a une solution optimale pour le problème étudié, soit la solution courante est fractionnaire et on ne peut plus ajouter de contraintes. Dans ce cas, la valeur de cette solution peut-être utilisée comme une borne supérieure (inférieure) dans une procédure de *Branch &Bound*. On choisit une variable fractionnaire x_i et on remplace le problème par deux sous-problèmes en fixant x_i à 0 dans l'un et à 1 dans l'autre. A chaque sous-problème, on associe notre borne supérieure (inférieure) ou, pour améliorer cette borne, on peut générer des contraintes violées pour chaque sous-problème et les ajouter au programme linéaire correspondant.

1.2.2. Flots et multiflots

Nous supposons que les fondements de la théorie des graphes sont connus, en particulier les notions de flot (A.2 page 144) et de multiflot (A.3 page 145). Nous allons voir les utilisations

de ce type de modélisation au travers de deux problèmes : le problème du flot de coût minimal et le problème du multiflot de coût minimal.

Problème de flot de coût minimal

Le **problème de flot de coût minimal** est un des problèmes les plus simples.

Trouver sur un réseau $G = (V, E)$ un flot f qui **minimise** une fonction coût $c(f)$ sous les contraintes

- f est un flot sur $\mathbb{Z}$, $\mathbb{Q}$ ou $\mathbb{R}$;
- f est compatible avec deux vecteurs capacités $(MIN_e)_{e\in E}$ et $(MAX_e)_{e\in E}$, vecteurs définis sur le même corps que f

Pl. 16 : Problème de flot de coût minimal

Si le coût $c(f)$ est **linéaire** — $c(f) = \sum_{e\in E} c_e f_e$ avec $c_e \in \mathbb{R}, \forall e \in E$ —le problème est **polynomial**. Il peut s'écrire comme un programme linéaire et peut être résolu par la recherche itérée de cycles de coûts moyens négatifs dans le graphe d'écart [GT89].

Si le coût $c(f)$ est **convexe**, le problème devient un problème d'**optimisation convexe**. On peut encore le résoudre en modifiant la procédure de résolution précédente en prenant comme coût de l'arc la dérivée de c [MIN81, MIN89, OMV00].

Si le coût $c(f)$ est **concave**, le problème est **NP-complet** [GSS80, MIN89]. Le problème est alors résolu avec des heuristiques.

Problème de multiflot de coût minimal

C'est un problème de base pour de nombreux problèmes d'optimisation dans les réseaux. Si $f = (f^k)^{k\in K}$ représente un multiflot, on note $Sum(f)$ le vecteur agrégé, c'est-à-dire que l'on a $Sum(f)_e = \sum_{k\in K} f_e^k, \forall e$.

Trouver sur un réseau $G = (V, E)$ un multiflot $f = (f^k)^{k\in K}$ qui minimise une fonction coût $c(f) = \sum_{e\in E} c_e(f)$
sous les contraintes

- f est un multiflot ;
- $Sum(f)$ est compatible avec $(MIN_e)_{e\in E}$ et $(MAX_e)_{e\in E}$, deux vecteurs capacités ;

Pl. 17 : Problème de multiflot de coût minimal

La nature de $c(f)$ conditionne, là encore, la difficulté du problème. Si chaque c_e est **linéaire**, le problème est linéaire. Malheureusement, celui-ci est souvent de très grande taille

et de forme particulière (entraînant un mauvais conditionnement ou une dégénérescence). De nombreuses méthodes sont mises en jeu : décomposition de Dantzig-Wolfe encore *appelée décomposition par les prix* [MIN86, ASS78, KEN78], application de la dualité Lagrangienne et algorithmes de sous-gradient [KS77], méthodes de points intérieurs [CL02, CG90, SM91]. Ce problème est **trivial** si les contraintes de compatibilité n'interviennent pas, il suffit de calculer le plus court chemin entre chaque couple origine-destination.

Si chaque c_e est **linéaire à coût fixe** (c'est-à-dire $c_e(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ \alpha_e + \beta_e x & \text{sinon} \end{cases}$ avec $(\alpha_e, \beta_e) \in \mathbb{R}^2$), les coûts sont représentatifs d'économies d'échelle. Malheureusement, le problème devient NP-complet [FH04a, Chapitre 5].

Si la fonction $c(f)$ est **convexe** et qu'il n'y a pas de **contrainte de capacité**, on peut envisager des méthodes comme la méthode de déviation de flot [FGK73]. Si, au contraire, il y a des contraintes de capacité, on peut envisager les différentes techniques, déjà citées, de l'optimisation convexe.

Pour un état de l'art plus précis sur les différentes méthodes de résolution, exactes ou heuristiques, on pourra se référer à [FH04a, FH04b].

1.2.3. Réseaux de Files d'Attente

Dans la plupart des formulations des problèmes de synthèse de réseaux, notamment celles à base de multiflot, on envisage les produits circulant sur le réseau comme étant des quantités moyennes. Ces modèles sont donc plutôt stratégiques et tactiques. Seul un système dynamique permet de valider certains indices de performance comme la qualité de service [MA03]. C'est à ce niveau qu'intervient le concept des files d'attente, le réseau est modélisé comme un réseau de files d'attente et on peut alors simuler le système pour juger de sa qualité. Les réseaux prennent en compte les taux de perte et de congestion ; la fonction de Kleinrock [KLE75] est une des fonctions les plus connues.

On pourra se référer à [FH04a, chapitres 2 et 3] pour l'utilisation des files d'attente dans les réseaux de communication.

1.2.4. Modèles de Théorie des Jeux

Ce sont les outils les plus fréquemment mis en jeu lorsqu'il est question de tarification sur les réseaux [BON63, OWE95]. Deux points de vue bien distincts s'opposent :

- la théorie des jeux coopératifs,
- la théorie des jeux non coopératifs.

La tarification vue au travers d'un modèle de **Jeu Coopératif** correspond à un processus d'imputation (et donc de décision) des coûts de la part d'un opérateur maître.

A l'opposé, les modèles de **Jeux Non Coopératifs** conçoivent la tarification comme le résultat d'un processus naturel d'ajustement des prix et des niveaux de production par des opérateurs concurrents confrontés à une demande élastique aux prix. C'est donc plus une optique prévisionnelle que décisionnelle. Les modèles conduisent le plus souvent à résoudre grâce au théorème du **point fixe** des équilibres. Un des **équilibres** les plus connus est celui de **Nash** [NAS51] : aucun des acteurs ne peut modifier sa décision sans voir dégrader son profit.

1.2.5. Simulation

La simulation est un outil particulier qui, s'il ne permet pas d'optimiser directement un système, est très utile pour évaluer ses performances. Les systèmes modélisés peuvent être très complexes et leur résolution analytique n'est souvent pas possible.

Dans le cadre de la synthèse de réseaux, deux types de simulation sont en compétition : la simulation à **événements discrets** *versus* la simulation **fluide** [FH04b]. Avec la simulation par événements discrets, on tente d'imiter le fonctionnement du système réel à un niveau de précision prédéterminé. Les flots (ou multiflots) sont modélisés de manière discrète : les produits circulant sur le réseau (clients, paquets, etc) sont des entités du modèle. Au contraire, le modèle fluide ne fait pas la distinction des "individus" qui transitent sur le réseau, il ne s'intéresse qu'aux changements de débit des différents flots à chaque point du réseau et permet de diminuer le nombre d'événements [ROS00].

Il existe des approches qui tentent d'allier la puissance des méthodes de résolution analytique à la précision de la modélisation de la simulation. Nous développerons ce point à l'annexe C page 161.

Pléthore de simulateurs existent pour les réseaux : *OPTNET, COMNET, NS2, Traffic...*

Les systèmes graphiques de données et la simulation visuelle viennent faciliter le processus d'aide à la décision ([BBEN97], logiciel GeoRoute/HASTUS par exemple [GIR])

1.3. Réseaux de mobilité et problèmes de tournées de véhicules

Nous allons nous intéresser maintenant à un problème de synthèse de réseaux sur un type particulier de réseaux : **les problèmes de synthèse de réseaux de mobilité**. Nous allons tout d'abord définir ce que l'on entend par **réseaux de mobilité**, puis nous passerons en revue les différents problèmes de tournées de véhicules qui sont très proches de ce type de problème.

1.3.1. Réseaux de mobilité

Différents objets circulent sur le réseau (biens, personnes, pièces, paquet d'informations, ...), éventuellement de **nature différente**. Ces objets, ou **produits**, doivent être acheminés d'une origine à une destination données. Il faut en router une certaine quantité et cette quantité peut dépendre de la qualité de service de l'acheminement proposé (on parle alors de demande élastique). Ces objets se déplacent à une certaine vitesse sur le réseau.

Pour aider à l'acheminement de ces produits, on dispose d'objets que l'on va qualifier de **conteneurs** (comme des véhicules : bus, train, avion, camion, ou une machine, etc). Ces conteneurs ont la propriété de se déplacer **plus vite** sur le réseau que les produits mais ils n'utilisent qu'un **sous-réseau** du réseau initial. On pourra éventuellement prendre en compte la nature de ces conteneurs (type, capacité, autonomie, ...) et certains temps de service (comme les temps de montée/descente, chargement/déchargement, temps d'attente).

L'objectif est de **fournir un système de transport**, c'est-à-dire, de planifier le déplacement des conteneurs afin que le routage induit des produits ait la meilleure qualité de service possible et ceci, avec un coût modéré, en d'autres mots, **contrôlé**.

Ce problème de transport se reformule naturellement en terme de synthèse de réseaux où cela revient à identifier un objet **infrastructure** sur lesquels se déplacent les conteneurs puis à **router les produits**.

Cette formulation est très proche des problèmes de tournées de véhicules, cas particuliers des problèmes de synthèse de réseaux, dont on se propose de voir maintenant les différents modèles et les méthodes de résolution qui leur sont propres. **La seule différence notoire avec les modèles de type tournées de véhicules "classiques" est que les produits ont une vie "propre" avant d'entrer dans le système de transport : ils peuvent bouger, seuls, avant et/ou après avoir éventuellement emprunté le système de transports.**

1.3.2. Le problème de Tournées de Véhicules "classique"

Cette section est consacrée à un problème de conception de tournées avec contraintes sur les nœuds, le Problème de Tournées de Véhicules (*Vehicle Routing Problem* ou *VRP*). Pour un exposé de tous ces problèmes, on pourra se référer à l'état de l'art de Bodin *et al* de 1983 [BGAB83], ou plus récemment à Toth et Vigo 2002 [TV02b].

Le problème de Tournées de Véhicules consiste, dans sa forme la plus simple, à optimiser les déplacements d'une flotte homogène de véhicules de capacité finie, à partir d'un dépôt unique. La flotte doit permettre de visiter un ensemble de clients pour un même type de service (dépôt ou collecte). L'optimisation peut se faire suivant différents critères : coût, temps, ... La première formulation de ce problème a été faite par Dantzig et Ramser dans les années 50 [DR59].

Outre le nom communément admis et déjà cité, ce problème est encore dénommé : *Vehicle Scheduling* [CW64, GAS67], *Vehicle Dispatching* [DR59, CE69] ou encore *Delivery Problem* [BQ64].

Le **problème classique** est le problème de tournées de véhicules avec capacité et une flotte homogène. (*Capacitated Vehicle Routing Problem* ou *CVRP*). Le CVRP est un problème NP-difficile au sens fort [LR81], c'est-à-dire que le problème n'est pas soluble en temps polynomial.

Formulation

Ainsi pour ce problème, on se place dans le cas où l'on veut, par exemple, desservir un ensemble de clients dont on connaît la demande. On cherche à effectuer les livraisons à un coût minimal et on dispose pour cela d'une flotte homogène de véhicules de capacité donnée (tous les véhicules sont identiques) située à un même dépôt. Chaque tournée débute et se termine au dépôt. Tous les clients doivent être desservis exactement une fois, c'est-à-dire que l'on n'envisage pas la préemption de la demande.

Soit $G = (V, E)$ un graphe orienté où l'ensemble de sommets $V = \{v_0, .., v_n\}$ représente le dépôt (sommet v_0) et les n clients (v_i pour $i \geq 1$). Par abus de notation, on utilisera indifféremment v_i et i pour le même sommet (client) i. E désigne l'ensemble des arcs du graphe. A chaque arc (i, j) est associé un coût c_{ij}. Une flotte de K véhicules tous identiques, de capacité Q, est stationnée au dépôt v_0. La demande d_i de chaque client i, $1 \leq i \leq n$, est connue. On note δ_j^i la fonction indicatrice qui vaut 1 si $i = j$ et 0 sinon.

Soit x_{ij}^k une variable binaire définie pour $i \neq j$, égale à 1 si le véhicule k passe au sommet v_j après être passé au sommet v_i et 0 sinon. Soit y_i^k une autre variable binaire qui vaut 1 si le véhicule k passe au sommet v_i et 0 sinon. On calcule aussi u_i^k qui est l'ordre dans lequel le véhicule k visite le sommet v_i.

Le programme linéaire (18) donne une formulation du problème CVRP, dite des trois indices, énoncée par Fisher et Jaikumar [FJ78, FJ81]. Elle se déduit directement de l'une

des premières formulations faite par Dantzig et al [DFJ54] pour le voyageur de commerce. Dans certains cas (réseau symétrique et inégalité triangulaire vérifiée), Laporte et al [LND85] ont montré qu'il était possible de formuler ce problème de manière plus compacte avec deux indices seulement.

Sur un graphe $G = (V, E)$ donné, trouver les tournées $x = (x_e^k)_{e\in E}^{k\in K}$ et les variables induites $y = (y_i^k)$ et $u = (u_i^k)$ qui **minimisent** $\sum_{k=1}^{K}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij}^k$

sous les contraintes :

$$\sum_{i=0}^{n} d_i y_i^k \leq Q \qquad \forall k \in \{1,..,M\} \tag{18.1}$$

$$\sum_{k=1}^{K} y_i^k = \delta_i^0(K-1)+1 \qquad \forall i \in \{0,..,n\}; \tag{18.2}$$

$$\sum_{i=0}^{n} x_{ij}^k = y_j^k \qquad \forall j \in \{0,..,n\}; \forall k \tag{18.3}$$

$$\sum_{j=0}^{n} x_{ij}^k = y_i^k \qquad \forall i \in \{0,..,n\}; \forall k \tag{18.4}$$

$$u_i^k - u_j^k + (n-1)x_{ij}^k \leq n-2 \qquad \forall (i,j) \in \{1,..,n\}^2, j > 1, i \neq j; \forall k \tag{18.5}$$

$$1 \leq u_i^k \leq n \qquad \forall i \in \{1,..,n\}; \forall k \tag{18.6}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in \{0,..,n\}^2, i \neq j; \forall k \tag{18.7}$$

$$y_i^k \in \{0,1\} \qquad \forall i \in \{0,..,n\}; \forall k \tag{18.8}$$

connaissant les coûts $(c_e)_{e\in E}$, la capacité Q des véhicules, et les demandes $(d_i)_{i\in V}$

Pl. 18 : Problème de Tournées de Véhicules avec Capacité (CVRP)

La fonction objectif vise à minimiser le coût total des tournées, le coût d'une tournée étant la somme des coûts des arcs utilisés. La première contrainte (18.1) impose le respect de la capacité des véhicules (tous les véhicules sont identiques). La contrainte (18.2) impose, à la fois, que tous les véhicules passent au dépôt et aussi que chaque client soit visité exactement une fois. (18.3) et (18.4) assurent du fait que si un véhicule k dessert le client i, il doit y arriver et en repartir ; elles imposent aussi que le véhicule k doit partir du dépôt et y revenir. Les variables u_i^k (18.6) sont des index topologiques, liés aux variables x_{ij}^k. Elles expriment l'ordre de passage des véhicules aux nœuds et permettent d'exprimer des contraintes d'exclusion de sous-tours (contraintes (18.5)), dites de Miller-Tucker-Zemlin [MTZ60]. Les dernières contraintes (18.6), (18.7) et (18.8) sont les contraintes triviales de définition des variables u_i^k, x_{ij}^k et y_i^k.

Il existe de nombreuses formulations plus performantes, notamment en ce qui concerne l'expression des contraintes d'exclusion des sous-tours : formulations de deux à quatre indices, énumération explicite des sous-tours comme pour le problème de voyageur de commerce

[DFJ54], *lifting* (renforcement) des contraintes (18.5) [DL91, KLB04], etc. L'expression de ces contraintes doit être choisie en fonction de son adéquation avec la méthode de résolution retenue [GP99]. Un des modèles les plus efficaces repose sur l'introduction d'un multiflot sur les véhicules [GG78]. Nous reparlerons en détail des techniques d'élimination de sous-tours à la section 2.4.4.

Il existe de nombreuses variantes et extensions de ce problème : flotte non-homogène, multidépôt, fenêtres de temps, préemption de la demande (voir [BGAB83, TV02b, GUE99] pour un énoncé des différentes variantes).

Méthodes de résolution exactes ou "presque" exactes :

Une méthode de résolution est dite **exacte** si elle garantit l'obtention d'une solution optimale du problème. Dans son article de 1992 [LAP92], Gilbert Laporte distingue trois grandes familles de méthodes exactes pour les problèmes de tournées de véhicule :

- les méthodes de recherche arborescente,
- la programmation dynamique [CEW71, CMT81],
- la programmation linéaire entière.

Les méthodes de **séparation et d'évaluation** (*Branch and Bound*) comptent parmi les plus efficaces pour résoudre de manière exacte ce problème. On pourra se référer à l'article de Toth et Vigo 2002 [TV02a] pour un état de l'art sur les méthodes exactes de résolution aussi bien en ce qui concerne la version symétrique que la version asymétrique : actuellement, il est possible de résoudre des instances avec une centaine de clients. Les méthodes de **coupes** (*Branch and Cut*) sont encore en plein développement [TV02b, chapitres 2 et 3].

Les formulations par **recouvrement** (*set-covering*), qui font appel à de la génération de colonnes, donnent aussi de bons résultats [LN87]. Fisher et Jaikumar [FJ81] utilisent une approche sur une triple indexation et décomposent le problème CVRP en un problème d'affectation généralisé et des problèmes de voyageurs de commerce.

Toutes ses méthodes exactes peuvent servir de base pour des résolutions "presque" exactes, différentes des heuristiques classiques dédiées aux problèmes de Tournées de Véhicules.

Méthodes de résolution heuristiques :

Au vu de la complexité du problème et des temps de calcul nécessaires, les heuristiques et par suite les métaheuristiques, sont utilisées de manière intensive pour obtenir rapidement de "bonnes" approximations sur des instances des problèmes.

Dans les états de l'art récents [LS98, CGL$^+$02], on trouve les heuristiques classiques utilisées pour la résolution du VRP. On distingue trois classes :

- les heuristiques de construction,
- les heuristiques 2-phases,
- les méthodes d'amélioration.

Les **heuristiques de construction** calculent en général un **coût d'insertion** des arcs. L'heuristique la plus connue est la méthode d'insertion de Clarke & Wright [CW64] mais on trouve également d'autres approches comme l'insertion séquentielle et les méthodes dites *matching based savings*.

On distingue trois types d'**heuristiques 2-phases** : "cluster first, route second", "route first, cluster second" et les algorithmes pétales. Pour la première famille, il s'agit de faire des sous-ensembles de sommets intéressants pour lesquels on construit ensuite une tournée (la méthode dite de balayage ou *sweep method*, en fait partie). En ce qui concerne la deuxième famille, certaines contraintes comme la taille des tournées sont relâchées. Une tournée hamiltonienne globale (problème du Voyageur de Commerce) peut être construite dans un premier temps. Celle-ci est ensuite cassée pour respecter les contraintes de taille. Les algorithmes pétales sont une extension des algorithmes de balayage : les tournées générées (pétales) sont sélectionnées en résolvant un problème de partionnement.

Les méthodes d'amélioration sont fondées sur des échanges d'un certain nombre d'arcs : on trouve d'abord les mouvements de Lin [LIN65, LK73] qui sont des k-opt et par la suite, tous les cas particuliers : 2-opt, 3-opt, λ-échange [OSM93], pour les problèmes classiques, Or-opt [OR76], 2-opt* [PR95] où l'ordre de la tournée n'est pas changé pour les problèmes avec fenêtres de temps. Tous les mouvements ont été initialement créés pour résoudre les problèmes de TSP et ont été ensuite adaptés pour le VRP.

Ces heuristiques ont été délaissées pour les métaheuristiques intrinsèquement plus puissantes :

- recherche tabou : cette métaheuristique a été développée indépendamment par Glover et Hansen en 1986 (voir Glover [GLO86, GLO89, GLO90, GL97], Hansen [HAN86] et Taillard [HTW97]),
- recuit simulé [KGV83, ČER85],
- algorithmes génétiques encore appelés évolutionnaires, introduits par Holland dès 1975 [HOL92]
- colonies de fourmis [CDM92, GTA99],
- Greedy Randomized Adaptative Search Procedure ou GRASP [FR89, FR95, FR],
- Variable Neighborhood Search ou VNS [MH97],
- la recherche par dispersion ou *Scatter Search*. [GLO77, GLO99]

Ces différentes métaheuristiques sont de plus en plus hybridées entre elles ou avec des méthodes exactes pour donner des schémas de résolution encore plus puissants.

1.3.3. Extensions diverses

Le problème central de tournées de véhicules sert de base à de nombreuses variantes. Parmi celles-ci, on peut citer :

- le problème de livraison-collecte,
- le problème de tournées de véhicules avec fenêtres de temps (cf [CDD+99] pour un état de l'art spécifique),
- le problème du ramassage scolaire,
- le problème de transport à la demande.

Le problème de conception de tournées de véhicules peut aussi être couplé avec des problèmes de planification, i.e. l'affectation des chauffeurs aux bus avec des horaires de travail (*scheduling problem*).

Dans le problème classique, on ne fait que livrer ou emporter un même type de marchandise. Dans l'extension du problème de **livraison-collecte** (*pickup and delivery problem*), on

peut réaliser les deux opérations en fonction du client. Il peut aussi y avoir des contraintes de service (vider le camion avant de collecter, par exemple). A titre d'exemples, on peut citer la (re)distribution du lait, les tournées postales...

Pour les problèmes de **tournées de véhicules avec fenêtres de temps**, la livraison ou la collecte de marchandises ne peut avoir lieu n'importe quand dans le temps : il faut respecter des fenêtres de temps pour le service, c'est-à-dire qu'il faut que le client soit servi entre deux dates butoir. Ces contraintes de temps peuvent être plus ou moins dures. En général, le véhicule peut arriver chez le client avant la date de service au plus tôt mais, il lui faut alors attendre cette date pour effectuer sa tâche. L'objectif du problème peut changer car on peut chercher à minimiser les temps d'attente de service.

Le problème du **ramassage scolaire** (*School Bus Routing Problem*) est une première approche de transport public. Il s'agit de déterminer des tournées de bus pour emmener des élèves à leur école (le matin) ou bien pour les déposer à proximité de leur domicile (le soir). Les critères d'optimisation de ce problème sont nombreux et souvent contradictoires. Du point de vue de la société de transport, il faut minimiser le nombre de tournées de bus à cause du coût d'achat puis de maintenance des bus. D'un autre coté, les tournées ne peuvent s'étaler en longueur : le conducteur ne doit pas conduire trop longtemps et un enfant ne doit pas passer trop de temps dans le bus. Il est même préférable que les tournées soient homogènes. Le remplissage du bus doit être équilibré et surtout, il ne faut pas que leur capacité soit dépassée. Si les élèves doivent marcher pour se rendre à leur arrêt de bus, il ne faut pas que cela soit trop long et qu'en moyenne, les élèves soient sur un pied d'égalité (il ne faut pas que ce soient toujours les mêmes élèves qui soient obligés de rester le plus longtemps dans bus ou qui doivent partir le plus tôt). En zone urbaine, un bus ne dessert en général qu'une seule école (exemple américain) et au vu de la densité de passagers, les tournées sont relativement courtes. Il est très difficile d'intégrer les contraintes d'accessibilité au système de transport, on suppose en général que les emplacements potentiels des arrêts de bus ont été préalablement fixés par une autorité régulatrice compétente. Il faut donc résoudre trois problèmes : un problème de localisation pour déterminer les arrêts de bus, un problème d'affectation sur les élèves pour leur attribuer un arrêt de bus et enfin un problème de tournées pour passer par tous les arrêts préalablement décidés. Il existe plusieurs heuristiques de résolution, la plus efficace, appelée *Allocation-Routing-Location*, consiste à découper des "zones" d'étudiants. Chacune de ces zones ne doit être desservie que par une seule tournée. Lors de la deuxième étape sont sélectionnés les arrêts de bus dans chaque zone. Il ne reste plus qu'à générer la tournée pour chaque zone.

Les problèmes de **Transport à la Demande** (*Dial-a-Ride Problem* ou *DARP*) [CL03] peuvent être vus comme une généralisation de tous les problèmes de tournées de véhicules déjà cités dans la mesure où ils incluent des fenêtres de temps et la livraison-collecte. Borndörfer [BOR98, chapitre 4] a utilisé une heuristique de type *Set Partitioning* pour une application pratique de transport de personnes à mobilité réduite à Berlin. De nombreuses villes possèdent un tel service mais l'optimisation est cruciale dans la mesure où les coût d'exploitation peuvent rapidement devenir prohibitifs. Le service de Berlin, très coûteux, a pu être sauvé grâce à ce travail d'optimisation. La problématique consiste généralement à déterminer les tournées du lendemain connaissant les besoins de déplacement des usagers. Au vu du public ciblé (personnes à mobilité réduite), il est impératif que le déplacement soit de **point à point**, c'est-à-dire de leur origine à leur destination et il faut considérer des temps de service pour la montée et la descente des véhicules.

Une autre facette de ces problèmes de tournées de véhicules (classique, livraison-collecte ou à la demande) est l'intégration d'une certaine souplesse dans la mesure où toutes les demandes ne sont pas connues au départ de la tournée mais elles évoluent au cours de celle-ci (suppression de demandes ou ajout de nouvelles). Ces nouvelles données nécessitent une réoptimisation et font appel à des algorithmes dédiés dits "**en ligne**".

Pour terminer ce tour d'horizon des différentes extensions et variantes du problème de tournées de véhicules, le tableau (1.1) rappelle un certain nombre de problèmes de tournées sur les nœuds du graphe.

ACVRP	Asymmetric Capacitated Vehicle Routing Problem
	Problème Asymétrique de Tournées de Véhicules avec Capacités
BRP	Bus Routing Problem
	Problème de tournées de bus
BTP	Bus Touring Problem
	Problème de l'excursion en bus
CVRP	Capacitated Vehicle Routing Problem
	Problème de Tournées de Véhicules avec Capacité
DARP	Dial-a-Ride Problem
	Problème de transport à la demande
MDVRP	Multi Depots Vehicle Routing Problem
	Problème de tournées de véhicules à dépôts multiples
MDVRPTW	Multi Depots Vehicle Routing Problem with Time Windows
	Problème de tournées de véhicules à dépôts multiples et Fenêtres de temps
OVRP	Open Vehicle Routing Problem
	Problème de Tournées de Véhicules Ouvert
PDVRP	Pickup and Delivery Vehicle Routing Problem
	Problème de Livraison-Collecte
MPDPTW	Multiple Pickup and Delivery Problem with Time Windows
	Problème avec Fenêtres de temps et livraisons-collectes multiples
SBRP	School Bus Routing Problem
	Problème du Ramassage Scolaire
SCVRP	Symmetric Capacitated Vehicle Routing Problem
	Problème Symétrique de Tournées de Véhicules avec Capacités
STSP	Selective Traveling Salesman Problem
	Problème du Voyageur de Commerce Sélectif
SVRP	Selective Vehicle Routing Problem
	Problème de Tournées de Véhicules Sélectives
SP	Scheduling Problem
	Problème de Planification (ou Planning)
TSP	Traveling Salesman Problem
	Problème du Voyageur de Commerce
VRP	Vehicle Routing Problem
	Problème de Tournées de Véhicules
VRPBTW	Vehicle Routing Problem with Backhauls and Time Windows
	Problème de Tournées de Véhicules avec Backhauls et Fenêtres de Temps
VRPSDC	Vehicle Routing Problem with Stochastic Demands and Customers
	Problème de Tournées de Véhicules avec Demandes et Clients Stochastiques
VRPTW	Vehicle Routing Problem with Time Windows
	Problème de Tournées de Véhicules avec Fenêtres de Temps

Tab. 1.1 – Abbréviations des différents problèmes

1.3.4. Synthèse

Le tableau (1.2) liste quelques problèmes de tournées de véhicules avec leur méthode de résolution. Nous avons utilisé les abréviations suivantes : S pour un état de l'art, H pour une résolution par heuristique, M pour une description de modèle. En ce qui concerne les méthodes, TS désigne la recherche Tabou, GA un algorithme génétique, B&B le Branch and Bound, SP le Set Partioning, CG la génération de colonnes et SA le recuit simulé.

Problème	Méthode(s)	Références
STPS	[S]	Feillet 2001 [FDG01]
BTP	[H]	Ladany 2000 [DL00]
BRP	[H] fourmis	Boryczka 2001 [BB01]
CVRP	[S] méthodes exactes	Laporte 1987 [LN87]
CVRP	[S] méthodes exactes	Vigo 2002 [TV02a]
CVRP	[H] B&B	Blasum 2000 [BH00]
SCVRP	[M] formulation multicommodité	Fischetti 1995 [FGT95]
VRP	B&B	Laporte 1985 [LND85]
VRP	[S] méthodes exactes	Laporte 1987 [LN87]
VRP	[S,H] TS	Rego 1994 [RR94]
VRP	[H] GA & TS	Duncan 1995 [DUN95]
VRP	[H] grande tournée divisée	Bowerman 1994 [BHC94]
VRP	[S] heuristiques classiques	Laporte 1998 [LS98]
VRP	[S]	Tan 2001[TLZO01]
VRP	[S, H] TS	Laporte 2001 [CGL$^+$02]
VRPTW	[H] TS	Potvin 1996 [PKGR96]
VRPTW	[S]	Cordeau 1999 [CDD$^+$99]
VRPTW	[S,H] SA, TS, GA	Tan 2001 [TLZO01]
VRPTW	[H] SP + CG	Liu 1998 [LLSC98]
VRPTW	[H] SA	Czech 2002 [CC02]
VRPTW		Simchi-Levi 1996 [BS96]
VRPTW		Simchi-Levi 1997 [BS97]
VRPTW		Madsen 1997 [KM97, FJM97]
VRPSD	[M]	Laporte 1993 [DLL93]
VRPSDC	[S,H] TS	Laporte 1996 [GLS96]
VRPBTW	[S,H] TS	Duhamel 1994 [DPR97]
MDVRPTW	[S,H] H 2-phases	Cheng 1998 [CR98]
MPDPTW		Lübbecke 2003[LÜB03]
PDPTW	[S,H] TS	Mitrovìc 2003[MKL03]
SBRP	[H]	Bodin 1983 [BGAB83]
SBRP	[H] LBH	Simchi-Levy 1994 [BBPS97]
SBRP	[H]	Bowerman 1995 [BHC95]
SBRP		Spasovic 2001 [SCK$^+$01]
SBRP	[S,H] 2 SA, 1 TS	Spada 2003 [SBL03]
BRP	[S,H]	Yan 2002[YC02]
DARP	[S]	Bodin 1983 [BGAB83]
DARP	[S]	Laporte 2003 [CL03]
DARP	[H] SP	Borndörfer 1998 [BOR98]

Tab. 1.2 – Récapitulatif des différents problèmes et méthodes

Dans ce chapitre, nous avons fait un état de l'art sur les problèmes de synthèse de réseaux tant du point de vue Télécommunications que du point de vue des Transports, que nous avons décrits plus précisément avec les différents problèmes de Tournées de Véhicules. Ce tour d'horizon des modèles a été complété par un survol des techniques utilisées pour leur résolution, aussi bien analytiques qu'heuristiques.

Nous avons aussi présenté le problème de transport qui est le sujet principal de cette thèse concernant les réseaux de mobilité, c'est-à-dire un problème de transport qui ne se réduit pas à un simple problème de Tournées de Véhicules mais qui est bien un problème de synthèse de réseaux dans la mesure où les produits que l'on doit acheminer, peuvent avoir une vie "propre" avant leur entrée dans le système de transport ou après leur sortie de ce système. Nous allons, au chapitre suivant, formaliser ce problème de transport.

Chapitre 2

Modèles pour les problèmes de synthèse de réseaux de mobilité avec demande élastique

Sommaire

Dans ce chapitre, nous allons donner quelques modèles pour le problème de synthèse de réseaux de mobilité avec demande élastique. De tels réseaux font intervenir deux classes d'objets circulant sur un réseau : les objets produits et les objets conteneurs. On cherche à effectuer un routage satisfaisant des produits en proposant un système de transport réalisé par les objets conteneurs.

Nous allons formaliser ce problème et donner un modèle général. Nous nous intéresserons ensuite à une instance simplifiée d'un tel modèle. Nous finirons ce chapitre par une sorte de "boîte à modèles" : nous envisagerons diverses extensions du modèle simplifié et nous donnerons au lecteur diverses pistes de modélisation pour des éléments que nous n'avons pas ou peu considérés. Nous finirons par quelques expérimentations sur quelques modèles.

Graphe :	
$G = (V, E)$	réseau support avec V : ensemble des nœuds E : l'ensemble des arcs
$E = A \cup \overline{A}$	E est partitionné en deux sous-ensembles A : ensemble des arcs "rapides" $\overline{A}$: ensemble des arcs "lents"
$V_A \subset V$	ensemble des nœuds qui sont extrémités d'un arc de A
$\omega^+(u)$	le cocycle sortant (arcs sortants) de $u \subset V$
$\omega^-(u)$	le cocycle entrant (arcs entrants) de $u \subset V$
Cardinalités :	
	$\lvert V\rvert = n$, $\lvert V_A\rvert = v$, $\lvert E\rvert = m$, $\lvert A\rvert = a$
Problème :	
$x = (x_e)_{e \in A}$	flot conteneur
$f = (f_e^k)_{e \in E}^{k \in K}$	multiflot produit (K commodités), pour $k \in K$ donné : origine o^k, destination d^k, demande D^k, temps induit t^k, satisfaction Φ^k

TAB. 2.1 – Lexique des différentes notations utilisées dans ce chapitre

2.1. Modèle général

On s'intéresse à un réseau $G = (V, E)$, où V est l'ensemble des nœuds et E l'ensemble des arcs, représentatif d'un système sur lequel peuvent circuler deux types d'objets (portion d'un tissu urbain, ferroviaire, aérien, une combinaison de ces derniers, une machine, ...). On considérera d'un côté, les **produits** (marchandises, pièces d'usinage ou personnes) que l'on cherche à acheminer d'un point du réseau à un autre et de l'autre des **conteneurs** (véhicule, navette, train, avion, bras transporteur...) qui ont la faculté de se déplacer plus vite sur le réseau. La vitesse de déplacement d'un produit peut être accélérée si ce produit se trouve dans un conteneur. **On cherche sur le réseau un ensemble de lignes**, noté Γ, représentant le déplacement des conteneurs pour que l'acheminement des produits induit par Γ soit optimal relativement à une certaine fonction de qualité de service.

Sauf mention contraire, nous faisons toujours référence aux notations données au tableau 2.1.

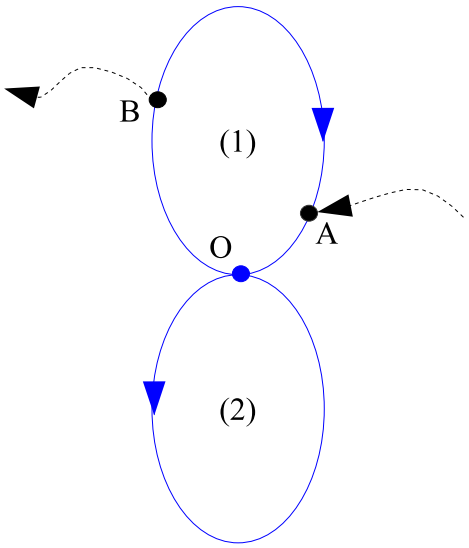
2.1.1. Flot conteneur

On suppose que les conteneurs, tous identiques, se déplacent sous forme de lignes sur un sous-ensemble d'arcs A de E. On pourra modéliser le déplacement des conteneurs grâce à un objet flot noté x, encore appelé **flot conteneur**.

La notion de flot permet en effet de représenter leur **déplacement** ainsi que le **nombre de conteneurs** nécessaires pour absorber le trafic sur le réseau. Une conséquence est que les conteneurs décrivent un ensemble de tournées fermées (dont le point de départ est aussi le point d'arrivée). Cette hypothèse n'est en aucun cas restrictive si l'on suppose que les objets conteneurs sont indissociables du point de vue des produits. En effet, un objet conteneur dont le temps de service est écoulé est implicitement remplacé par un autre de même nature et le produit ne fait pas de différence entre les deux conteneurs.

Identifier le déplacement des conteneurs à un flot permet aussi d'intégrer facilement la notion de **capacité** des conteneurs. Si l'on appelle Q cette capacité, un arc e de A dont le flot est x_e pourra transporter au plus $x_e.Q$ produits.

Ainsi la modélisation par flot de notre objet inconnu est très intéressante mais elle ne permet pas de prendre en compte tous les problèmes liés au routage de produits dans un réseau. En effet, elle ne permet qu'**un support limité de la problématique temps**. En particulier, on ne peut modéliser efficacement le problème des connexions complexes. La figure 2.1 illustre une limitation de cette interprétation.



Considérons un produit entrant dans la tournée au point A et en sortant au point B. La modélisation par flot ne permet pas de lever l'ambiguïté de comportement au point O, point de connexion entre les deux boucles d'une même ligne : le produit passe-t'il dans la deuxième boucle ou pas ?

FIG. 2.1 – Illustration des limitations de la modélisation par un flot

2.1.2. Multiflot produits induit

Les produits circulent sur le réseau à deux vitesses suivant le type des arcs qu'ils empruntent : à **vitesse "lente"** sur les arcs de $\overline{A}$ et à **vitesse rapide** sur les arcs de A. On impose la condition supplémentaire qu'un produit ne se déplace à la vitesse rapide que s'il se trouve dans un conteneur (contraintes dites de couplage). On dispose de K familles de produits et chaque type de produit k est caractérisé par la donnée d'un triplet (o^k, d^k, D^k) où $o^k \in V$ représente l'origine du déplacement, $d^k \in V$ la destination et D^k la demande associée, c'est-à-dire le nombre de produits de ce type à acheminer. La terminologie usuelle est d'appeler **commodité** un tel triplet. De cette manière, on pourra représenter le routage des produits sous la forme d'un multiflot f, encore appelé **multiflot produits**, $f = (f_e^k)_{e\in E}^{k\in K}$ associé au K commodités. f_e^k représentera la contribution du produit k à l'arc e.

La loi de conservation pour chaque commodité s'appliquera à tous les nœuds distincts de l'origine et de la destination si l'on suppose qu'aucun produit ne se perd.

On considère dans la suite de ce document que le sous-graphe $(V_{\overline{A}}, \overline{A})$ de G est **fortement connexe**, c'est-à-dire qu'il est toujours possible de router les produits de o^k vers d^k en prenant un chemin composé uniquement d'arcs parcourus à la vitesse lente.

2.1.3. Fonction objectif : évaluation du routage et qualité de service

L'objectif de notre modèle est d'avoir la meilleure qualité de service possible (ou *Quality of Service*, QoS). On fait l'hypothèse que la qualité de service totale est une fonction se décomposant comme la somme des qualités de service pour chaque produit QoS^k et dépendante du routage induit par le système proposé. Ainsi la qualité de service globale s'écrit :

$$QoS(f) = \sum_{k\in K} QoS^k(f^k)$$

On pourra s'intéresser au cas où la demande d'un produit est **élastique**, c'est-à-dire fonction de la qualité de service proposée par le système de transport.

2.1.4. Modèle général

Ainsi, on peut exprimer notre problème sous la forme suivante :

Trouver sur un réseau $G = (V, E)$, un flot conteneur $x = (x_e)_{e \in A}$ et un multiflot produit $f = (f_e^k)_{e \in E}^{k \in K}$ tels que :

- f^k route la quantité D^k de l'origine o^k à la destination d^k
- x respecte des contraintes topologiques ;
- x respecte des contraintes budgétaires ;
- les contraintes de couplage de f avec x soient respectées (contraintes de support) ;
- la qualité de service $\sum_{k \in K} QoS^k(f^k)$ est **maximale** ;

connaissant $A \subset V$ et des commodités $(D^k, o^k, d^k)_{k \in K}$

Pl. 19 : Modèle général d'un problème de synthèse de réseaux de mobilité avec demande élastique

Un tel modèle est encore trop général pour pouvoir être étudié. Nous allons donc tout d'abord nous intéresser à une instance très simple. Nous donnerons ensuite quelques extensions à ce modèle simplifié. La **première motivation** est de proposer **différentes alternatives dans le cadre d'une aide à la décision** en fonction des objectifs recherchés, tel un jeu de construction.

Nous cherchons aussi à obtenir des modèles qui soient directement exploitables par des solveurs de programmes linéaires comme CPLEX, XPRESS ou GLPK, même si nous explorons des pistes qui sont clairement non linéaires ou font intervenir d'autres mécanismes (multiobjectif par exemple). On se servira donc du formalisme issu de la programmation linéaire en nombre entier. Le **second objectif** est de pouvoir disposer d'un **panel de solutions** lorsque cela est possible pour servir de **référence** et ainsi permettre l'évaluation de la qualité des métaheuristiques que nous avons adaptées pour cette famille de problèmes.

2.2. Un cas simple : un routage satisfaisant au niveau temps

Ce cas simple du modèle général se concentre sur les **temps de déplacement** des produits. On mesure la qualité de service en fonction du temps d'acheminement correspondant de l'origine à la destination de la commodité k. On appelle Φ^k la fonction de qualité de service dépendante du temps. Elle sera en général paramétrée par différents temps de déplacement, dits de référence, qui serviront à établir sa forme, suivant sa nature.

2.2.1. Fonction objectif

On note S la fonction objectif du problème, i.e. la satisfaction totale. Si l'on appelle t_Γ^k **le temps de parcours de la commodité** k **induit par le système de transport** Γ, on a :

$$S = QoS = \sum_{k \in K} D^k \Phi^k(t_\Gamma^k(f^k))$$

Par abus de notation, on notera également S comme une fonction directement du multiflot f et non pas du temps induit, c'est-à-dire : $S = \sum_{k \in K} D^k \Phi^k(f^k)$

2.2.2. Contraintes sur le système de transport recherché (conteneurs)

Nous rappelons que nous cherchons à déterminer le système de transport Γ, un flot représentant le déplacement des conteneurs. Plusieurs contraintes peuvent être associées à ce flot :

- les contraintes de **conservation de flot** tout d'abord, qui s'écrivent naturellement :

$$\sum_{e \in \omega^-(u)} x_e - \sum_{e \in \omega^+(u)} x_e = 0, \forall u \in V_A$$

- on peut vouloir ensuite contrôler le coût du système de transport et donc, si l'on dispose d'une famille de coût $(p_e)_{e \in A}$ et d'une borne supérieure $p_{\max}$ de ce coût, on a la **contrainte budgétaire** suivante :

$$\sum_{e \in A} p_e x_e \leq p_{\max}$$

- enfin, on peut tout à fait prendre en compte des **contraintes de capacité** sur les arcs, c'est-à-dire sur le nombre de conteneur circulant sur un arc, et ainsi, si $(MAX_e)_{e \in A}$ est une famille de bornes positives, on a :

$$0 \leq x_e \leq MAX_e, \forall e \in A$$

2.2.3. Contraintes sur le routage induit des produits

De cet objet principal découle le routage des produits que l'on modélise sous la forme d'un multiflot. Pour simplifier la modélisation, on prend un **multiflot fractionnaire** à valeurs dans l'intervalle réel $[0, 1]$. Pour connaître la quantité de produit k sur un arc e, il suffit de multiplier f_e^k par la demande D^k associée à la commodité. Celui-ci doit donc respecter les contraintes de conservation du flot pour chaque composante :

$$\sum_{e \in \omega^-(u)} f_e^k - \sum_{e \in \omega^+(u)} f_e^k = b_u^k, \forall u \in V, \forall k$$

où b_u^k est une constante dépendante de la nature du sommet u considéré pour la commodité k. Cette constante vaut -1 si v est l'origine o^k de la commodité, $+1$ s'il s'agit de la destination d^k, et 0 pour tous les autres sommets. On peut vouloir calculer le temps de déplacement induit

par le système de transport pour une commodité donnée $k \in K$. Connaissant les temps de parcours c_e des arcs de G, ce temps s'écrit naturellement comme suit :

$$t_\Gamma^k = \sum_{e \in E} c_e f_e^k$$

Par abus de notation, on le notera le plus souvent t^k.

2.2.4. Contraintes de couplage

Les contraintes de couplage sont au cœur du problème que nous étudions. Elles symbolisent la relation forte entre le routage des produits et le flot des conteneurs qui circulent sur le réseau. Comme dans tous les problèmes de synthèse de réseaux, ce sont ces contraintes qui rendent le **problème difficile**.

Elles se déclinent de plusieurs façons, suivant les paramètres que l'on veut prendre en compte ou encore suivant les méthodes de résolution (certaines contraintes peuvent avoir un meilleur comportement vis-à-vis des différentes relaxations comme on a pu le constater lors de l'étude des différents problèmes au chapitre consacré à l'état de l'art).

Conteneurs de taille infinie

Supposons tout d'abord que la capacité des conteneurs est infinie (ce qui peut être le cas par exemple sur un réseau si, sur une échelle de temps donnée, la fréquence des conteneurs est suffisamment élevée pour absorber tout le trafic). On peut normaliser les demandes des produits, $\sum_{k \in K} D^k = 1$ et alors les contraintes de couplage sont :

$$\sum_{k \in K} f_e^k \leq x_e, \forall e \in A$$

Ce qui donne m contraintes de couplage.

On peut très bien choisir d'éclater cette contrainte au niveau des commodités, et plutôt choisir de l'écrire :

$$f_e^k \leq x_e, \forall e \in A, \forall k \in K$$

On a alors mK contraintes. Mais cette formulation peut être plus forte en terme de relaxation.

Conteneurs identiques de taille donnée

Supposons maintenant que la capacité des conteneurs, tous identiques, est finie et fixée à la valeur Q. La contrainte de couplage entre le flot et le multiflot, s'écrit naturellement :

$$\sum_{k \in K} D^k f_e^k \leq Q x_e, \forall e \in A$$

2.2.5. Modèle

Voici donc une instance simple du problème de synthèse de réseaux de mobilité avec demande élastique où les conteneurs sont de capacité infinie et où le support temps est limité :

Trouver sur un réseau $G = (V, E)$, un flot conteneur $x = (x_e)_{e \in A}$ et un multiflot produit $f = (f_e^k)_{e \in E}^{k \in K}$ tels que $S = \sum_{k \in K} D_k \Phi^k(f^k)$ soit **maximale** (20.0)

sous les contraintes :

$$\sum_{e \in A} p_e x_e \leq p_{\max} \tag{20.1}$$

$$\sum_{e \in \omega^-(u)} x_e - \sum_{e \in \omega^+(u)} x_e = 0 \qquad \forall u \in V_A \tag{20.2}$$

$$\sum_{e \in \omega^-(u)} f_e^k - \sum_{e \in \omega^+(u)} f_e^k = b_u^k \qquad \forall u \in V, \forall k \tag{20.3}$$

$$f_e^k \leq x_e \qquad \forall e \in A, \forall k \tag{20.4}$$

$$x_e \in \mathbb{N}, 0 \leq x_e \leq MAX_e \qquad \forall e \in A \tag{20.5}$$

$$f_e^k \in [0, 1] \qquad \forall e \in E, \forall k \tag{20.6}$$

connaissant $A \subset E$, des commodités $(D^k, o^k, d^k)_{k \in K}$, des coûts $p_{\max}$, $(p_e)_{e \in A}$ et $(c_e)_{e \in E}$, et des capacités $(MAX_e)_{e \in A}$

Pl. 20 : Instance simple du problème de synthèse de réseaux de mobilité

En raison de la problématique soulevée en introduction qui consiste à être capable de connaître des fonctions de qualité de service acceptables et réalistes, nous avons laissé une forme encore très générale dans cette formulation. Nous allons choisir des expressions particulières de ces fonctions au point 2.3.

Ce modèle se place dans l'optique où l'on cherche des lignes de transport associées aux conteneurs. Avec une modélisation par flot, **ces lignes sont fermées**, on peut donc aussi parler de circuits ou encore de tournées. Un tel modèle ne permet pas de connaître, a priori, ni le nombre de lignes créées ni leurs qualités intrinsèques (forme, taille, ...). Sans analyse, on ne sait pas non plus si des connexions complexes sont présentes sur le réseau. Nous verrons comment ajouter des contraintes pour fixer le nombre de tournées désirées au point 2.4.6.

Comme nous l'avons déjà évoqué, ce modèle ne prend pas en compte certains problèmes liés à la **gestion du temps** : **synchronisation des lignes** et **connexions complexes**. Nous pouvons juste supposer qu'il y a suffisamment de conteneurs (au niveau de la fréquence) pour absorber la demande globale.

Le déplacement des conteneurs se fait sur les arcs rapides sur des lignes déterminées par x. Le déplacement des produits se fait suivant **la règle du plus court chemin** (PCC) de l'origine à la destination du trajet. Dans certains cas, cette modélisation n'est pas appropriée, nous verrons comment changer le **mode de déplacement** au point 2.4.8.

2.2.6. Complexité spatiale

Variables	
$\mathbb{N}$	a
$\mathbb{R}$	Km

Contraintes	
égalités	$Kn + v$
"inégalités"	$K(2m + a + 1) + 2a + 2$
	$K(2m + a) + 2a + 1$

Le nombre de variables entières, le nombre d'arcs rapides a, peut devenir problématique lors de la résolution d'un problème.

2.2.7. Complexité théorique

Montrons que ce problème, même avec une formulation aussi simple, est un **problème NP-difficile**.

Plaçons-nous dans les conditions suivantes : $A = E$, $\Phi^k(f^k) = \sum_e c_e f_e^k$, $D_k = \frac{n(n-1)}{2}$. On cherche alors à minimiser $\sum_e x_e$. Chercher une telle somme inférieure ou égale à n ($n = |V|$) revient à trouver un cycle hamiltonien et se ramène donc à un problème de Voyageur de Commerce qui est NP-difficile [LR81], même si quelques unes des instances de ce problème peuvent être résolues en temps polynomial [LAP92]. ♦

2.2.8. Borne supérieure sur la fonction objectif

Nous aimerions répondre à la question de savoir si on peut disposer d'une bonne borne sur l'évaluation de la fonction objectif S.

Naturellement, si la contrainte (20.1) est relâchée, il est très facile de calculer une borne supérieure $S_{\max}$ de S. En effet, elle est obtenue si toutes les commodités sont routées avec une qualité de service optimale et vaut :

$$S_{\max} = \sum_{k \in K} D^k$$

Bien entendu, dans le cas général, cette borne est grossière et doit être modérée suivant les contraintes sur le coût du système de transport Γ , notamment (20.1). Nous ne disposons pas d'une borne plus fine qui s'exprimerait en fonction de ce coût. Cependant, on peut espérer obtenir une bonne estimation en résolvant un programme linéaire où les contraintes d'intégrité du flot sont relâchées : en effet, la satisfaction réelle ne peut être supérieure à la satisfaction obtenue en relâchant ces contraintes. Nous discutons du *gap*, écart d'évaluation de la fonction objectif entre la solution entière et la solution obtenue par relaxation, à la section 2.5.

2.3. Demande élastique et fonction de satisfaction

Nous avons pour l'instant supposé que le routage des produits se faisait sur des critères de qualité de service. Nous voulons étudier le cas où la demande des produits est **élastique**, c'est-à-dire qu'elle varie en fonction de l'offre que le "produit" perçoit au travers de la qualité de service. En toute rigueur, une telle demande — variant selon un mode particulier — devrait être qualifiée de *variable*, mais comme nous nous intéressons exclusivement à sa variation suivant le facteur temps, nous pouvons bien dire qu'elle est **élastique** [OPP95].

Intéressons-nous maintenant à une commodité, i.e. à un besoin de routage d'un produit sur le réseau. On peut considérer que la demande peut s'exprimer sous la forme d'un produit d'une demande nominale (maximale) par une certaine fonction Φ dépendante de la satisfaction. Cette satisfaction sera bien entendu dépendante de la qualité de service proposée.

$$\text{Demande}_{\text{effective}} = \text{Demande}_{\text{nominale}} \,.\, \Phi(QoS)$$

On ne connaît que fort **peu de choses** sur la fonction de pondération : cette fonction est croissante avec la qualité de service. Elle est proche de zéro si la qualité de service est médiocre, et maximale — proche de 1 — si celle-ci est excellente (figure 2.2).

Nous allons donner quelques fonctions de satisfaction (ou d'acceptation) que nous avons envisagées. En abscisse, on place la perception X du trajet par le produit (usager, ou personne qui commande le déplacement d'une pièce ou d'un bien) et, en ordonnée, la satisfaction relative ou encore le taux d'acceptation. On dira que le routage est satisfaisant si la perception est proche de zéro et de moins en moins satisfaisant si cette valeur augmente. Nous avons choisi cette échelle car elle est particulièrement adaptée à la perception du routage suivant le temps de parcours. La figure 2.3 est une fonction relativement réaliste avec un comportement nuancé sur la portion décroissante.

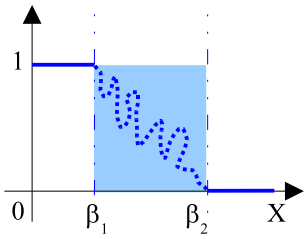


FIG. 2.2 – Fonction de satis. réelle

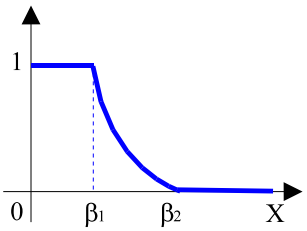


FIG. 2.3 – Fonction de satis. réaliste

Cette fonction peut être approximée grossièrement par une fonction linéaire par morceaux (figure 2.4). Mais, dans certains cas, on est amené à utiliser des fonctions encore plus simples. La figure 2.5 est une fonction barrière : l'usager emprunte le système de bus ou ne l'emprunte pas.

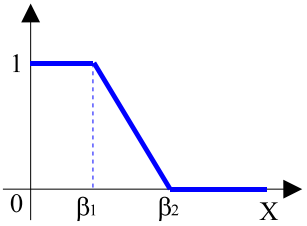


FIG. 2.4 – Fonction de satis. linéaire par morceaux

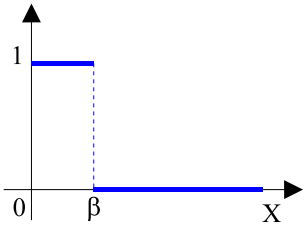


FIG. 2.5 – Fonction de satis. barrière

Nous ne disposons pas d'études ou de données pour proposer des fonctions de satisfaction réelles. Le choix d'une forme de fonction de satisfaction par rapport à une autre est guidé, dans la suite de ce document, par le choix de la méthode de résolution. On se contentera des

formes linéaires "simples" — modèles (2.4) et (2.5) — pour l'expérimentation des modèles linéaires. Les résolutions avec les heuristiques permettent l'usage de fonctions de satisfaction complexes.

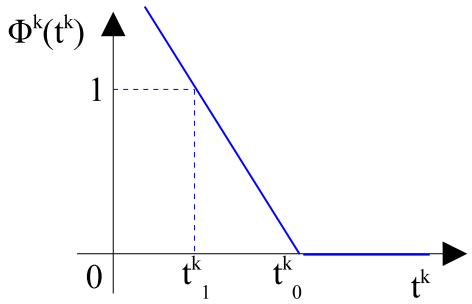
2.4. Quelques extensions

Dans cette section, nous allons donner des modélisations pour prendre en compte certains paramètres. Sauf mentions contraires, il suffit de les injecter dans le modèle simple. Certaines modélisations sont incompatibles entre elles. Certaines autres sont aussi très difficiles à mettre en œuvre, soit parce que la complexité spatiale explose, soit parce qu'elles composent avec d'autres techniques d'optimisation, comme l'optimisation multiobjectif par exemple.

2.4.1. Qualité de Service : une fonction simple

Dans ce paragraphe, nous allons étudier l'impact de la forme de la fonction de satisfaction sur le routage. Toutes les commodités utilisent le même type de fonction. Nous nous limitons aux fonctions affines par morceaux afin de pouvoir les injecter directement dans un solveur de programme linéaire.

La fonction de satisfaction est paramétrée par deux temps de référence : t_0^k et t_1^k. t_0^k représente le temps à partir duquel la satisfaction est nulle et t_1^k le temps maximal pour lequel la satisfaction est égale à 1.



L'expression associée est :

$$\Phi^k(t^k) = \begin{cases} -\frac{1}{t_0^k - t_1^k} t^k + \frac{t_0^k}{t_0^k - t_1^k} & \text{si } t^k \leq t_0^k \\ 0 & \text{si } t^k > t_0^k \end{cases}$$

FIG. 2.6 – Fonction de satisfaction affine

Comment intégrer cette équation dans un programme linéaire ?

Là encore, tout dépend de ce que l'on cherche à faire, les valeurs extrêmes t_0^k et t_1^k ne posent pas de problème si elles sont bien choisies. Si t_0^k est un temps au moins égal au temps de plus court chemin en vitesse lente, ce temps n'intervient plus. Dans le cas contraire, il faut ajouter au programme linéaire la contrainte :

$$t^k \leq t_0^k \quad \text{ou encore} \quad \sum_{e \in E} c_e f_e^k \leq t_0^k$$

On peut encore estimer qu'obtenir une satisfaction négative n'est pas un problème en soi et que le routage sur certains produits est tellement mauvais que cela pénalise l'objectif global. En ce qui concerne le temps t_1^k, on peut éventuellement admettre une satisfaction supérieure à l'unité. Dans le cas contraire, il faut prendre un temps inférieur au temps d'un plus court chemin en vitesse rapide.

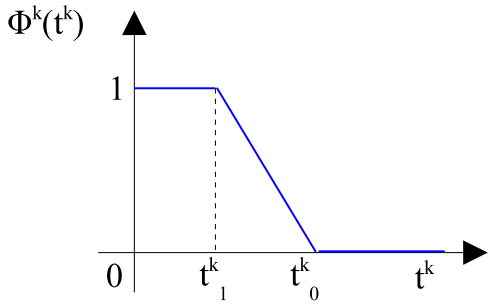
Et ainsi, la fonction de satisfaction globale a pour expression :

$$S = \sum_{k \in K} D^k \Phi^k(t^k) = -\sum_{k \in K} \frac{D^k}{t_0^k - t_1^k} t^k + \sum_{k \in K} \frac{D^k t_0^k}{t_0^k - t_1^k}$$

Nous ne donnerons au solveur linéaire que la partie non constante de l'expression de la satisfaction. Il suffit de remplacer la fonction objectif du modèle PL.20 par celle-ci.

2.4.2. Qualité de Service : une fonction linéaire plus réaliste

La fonction de satisfaction que nous décrivons maintenant est plus proche d'une fonction réaliste, même si elle est affine par morceaux. Elle présente deux paliers, l'un pour la satisfaction maximale, l'autre la satisfaction minimale. La portion entre les deux est linéaire.



L'expression associée est :

$$\Phi^k(t^k) = \begin{cases} 1 & \text{si} \quad t \leq t_1^k \\ \dfrac{t_0^k - t^k}{t_0^k - t_1^k} & \text{si} \quad t^k \in [t_1^k, t_0^k] \\ 0 & \text{si} \quad t^k \geq t_0^k \end{cases}$$

FIG. 2.7 – Expression de $\Phi^k(f^k)$

Une telle fonction est plus difficile à modéliser avec le formalisme de la programmation linéaire. Appelons $q^k = D^k.\Phi^k(f^k)$ la contribution de la commodité k à la fonction objectif, c'est-à-dire le nombre de produits effectivement routés pour la commodité k. Naturellement, on a les inégalités suivantes : $0 \leq q^k \leq D^k, \forall k$. La fonction objectif se simplifie en $\sum_{k \in K} q^k$.

Changement d'origine

Essayons donc maintenant d'exprimer q^k suivant les deux valeurs de référence t_0^k et t_1^k. Nous allons choisir arbitrairement t_0^k comme valeur de référence et nous allons translater (**changement d'origine**) toutes les quantités t^k et q^k par rapport à cette valeur.

On a donc $t^k = t_0^k + t_+^k - t_-^k$ avec t_+^k et t_-^k positifs, mais non tous positifs strictement simultanément, c'est-à-dire $t_+^k.t_-^k = 0$

Faisons de même pour q^k. q^k s'écrit $q^k = q_+^k + q_-^k$ où q_-^k est la contribution lorsque $t^k \leq t_0^k$ et q_+^k est la contribution lorsque $t^k > t_0^k$. Bien entendu, cette dernière est nulle et donc, pour simplifier les notations, on garde l'écriture q^k pour la quantité de produits circulant.
En résumé, avec la nouvelle origine, la quantité de produit circulant s'écrit simplement :

$$\begin{cases} 0 \leq q^k \leq D^k \\ q^k \leq \frac{t_-^k}{t_0^k - t_1^k} \end{cases}$$

Linéarisation des contraintes d'exclusion de signe entre t_-^k et t_+^k

Cependant, il reste un problème de taille si l'on veut exploiter directement un tel modèle avec un solveur de programmes linéaires : les contraintes de changement d'origine $t_+^k.t_-^k = 0$ qui ne sont **pas linéaires**.

Les solveurs usuels sont tous capables de faire de l'optimisation quadratique (enfin quand seule la fonction objectif est quadratique). La première idée consiste à relâcher ces contraintes non linéaires et à les introduire dans la fonction objectif. La fonction est alors de la forme $\sum_{k \in K} q^k - M \sum_{k \in K} (t_-^k.t_+^k + t_+^k.t_-^k)/2$ où M est une constante suffisamment grande (gardons à l'esprit que le produit doit être nul).

Malheureusement, le terme quadratique ainsi défini n'est pas semi-défini positif et l'optimisation quadratique n'est pas possible.

En effet, considérons la matrice qui représente toutes les variables q^k puis tous les couples (t_+^k, t_-^k), cette matrice est diagonale par bloc. Le premier bloc est une matrice identité, les autres blocs sont de la forme $((0, 1/2)(1/2, 0))$ et ont pour valeurs propres $-1/2$ et $1/2$. Toutes les valeurs propres de la matrice ne sont pas positives donc la matrice n'est pas semi-définie positive.♦

La deuxième idée est de prendre une nouvelle variable binaire pour chaque commodité. Cette variable nous renseignera sur la quantité — de t_+^k ou t_-^k — qui est non nulle. Soit b^k une telle variable, on veut alors :

$$\begin{cases} b^k = 0 \Rightarrow t_-^k = 0 \\ b^k = 1 \Rightarrow t_+^k = 0 \end{cases}$$

Il faut donc générer les contraintes suivantes en introduisant une constante suffisamment grande M^k :

$$\begin{cases} t_-^k - t_0^k b^k \leq 0 \\ t_+^k + M^k b^k \leq M^k \end{cases}$$

On peut prendre comme constante M^k toute valeur supérieure au PCC en vitesse lente pour la commodité.

Si nous avons réussi, de cette manière, à linéariser les contraintes d'exclusion de signe entre t_+^k et t_-^k, il n'en reste pas moins que cette méthode est inélégante dans la mesure où elle rajoute K variables binaires au problème. Elle fait donc exploser la complexité spatiale du problème et on ne peut s'attendre qu'à de bien piètres résultats avec un solveur de programmes linéaires lors de processus de recherche arborescente de la solution.

Nous pouvons aussi remarquer que la famille de contraintes que l'on a rajoutée a un très mauvais comportement lorsque le problème est relâché. Celui-ci est d'autant plus marqué que M^k est grand : il est difficile de fermer le *gap* entre la solution entière réalisable obtenue et la solution obtenue par relaxation.

Notons enfin que pour des valeurs bien choisies de t_0^k et t_1^k, il n'est pas nécessaire d'utiliser un tel modèle mais que le modèle avec la fonction de satisfaction simple est suffisant.

Finalement, le modèle linéaire est le suivant :

Trouver sur un réseau $G = (V, E)$, un flot conteneur $x = (x_e)_{e\in A}$ et un multiflot produit $f = (f_e^k)_{e\in E}^{k\in K}$ tels que $\sum_{k\in K} q^k$ soit **maximale**
sous les contraintes

$$\sum_{e\in A} p_e x_e \leq p_{\max} \tag{21.1}$$

$$\sum_{e\in\omega^-(u)} x_e - \sum_{e\in\omega^+(u)} x_e = 0 \qquad \forall u \in V_A \tag{21.2}$$

$$\sum_{e\in E} c_e f_e^k - t_0^k + t_-^k - t_+^k = 0 \qquad \forall k \in K \tag{21.3}$$

$$t_-^k - t_0^k b^k \leq 0 \qquad \forall k \tag{21.4}$$

$$t_+^k + M^k b^k \leq M^k \qquad \forall k \tag{21.5}$$

$$\sum_{e\in\omega^-(u)} f_e^k - \sum_{e\in\omega^+(u)} f_e^k = b_u^k \qquad \forall u \in V, \forall k \tag{21.6}$$

$$q^k - 1/(t_0^k - t_1^k).t_-^k \leq 0 \qquad \forall k \tag{21.7}$$

$$0 \leq q^k \leq D^k \qquad \forall k \tag{21.8}$$

$$f_e^k \leq x_e \qquad \forall e \in A, \forall k \tag{21.9}$$

$$0 \leq x_e \leq MAX_e,\ x_e \in \mathbb{N} \qquad \forall e \in A \tag{21.10}$$

$$f_e^k \in [0, 1] \qquad \forall e \in E, \forall k \tag{21.11}$$

où q^k, t_-^k et t_-^k sont des variables intermédiaires ;
connaissant $A \subset E$, des commodités $(D^k, o^k, d^k)_{k\in K}$, des coûts $p_{\max}$, $(p_e)_{e\in A}$ et $(c_e)_{e\in E}$, et des capacité $(MAX_e)_{e\in A}$

Pl. 21 : Modélisation avec une fonction affine par morceaux plus réaliste

Variables	
$\mathbb{N}$	$a + K$
$\mathbb{R}$	$K(m + 3)$

Contraintes	
égalités	$K(n + 1) + v$
"inégalités"	$K(a + 2m + 5) + 2a + 1$
	$K(n + a + 2m + 6) + v + 2a + 1$

Là encore, le nombre de variables "à problème" (les variables entières) est important : il est fonction du nombre d'arcs du graphe et surtout du nombre de commodités.

Revenons maintenant sur la contrainte de couplage (21.9). La contrainte donnée dans le PL ne prend pas en compte la notion de capacité des conteneurs. Si on veut la prendre en compte, avec une flotte homogène, il faut l'écrire, avec nos notations :

$$\sum_{k\in K} q^k.f_e^k \leq Qx_e, \forall e \in A$$

Sous cette forme, elle n'est malheureusement pas linéaire. Mais il suffit de substituer aux variables de multiflot (f_e^k) des quantités (q_e^k) où q_e^k représente finalement la contribution — le nombre de produits — de l'arc e à la commodité k. On a alors les relations : $q^k = \sum_{\omega^+(o^k)} q_e^k$ et des contraintes de couplage, linéaires, de la forme :

$$\sum_{k \in K} q_e^k \leq Q.x_e, \forall e \in A$$

En conclusion, nous avons formulé notre problème avec une fonction de satisfaction encore linéaire mais bien plus complexe. La complexité spatiale de ce problème, et en particulier le nombre de variables entières, en font un problème impossible à résoudre actuellement avec des outils de résolution de programmes linéaires comme CPLEX.

2.4.3. Qualité de service : analogie avec le problème du diamètre

Nous avons supposé au début de ce chapitre que la fonction de satisfaction globale était **séparable** suivant les différentes commodités et pouvait donc s'écrire comme une somme des différentes qualités de service.

Nous supposons dans ce paragraphe que ce n'est plus le cas et que **la qualité de service de l'ensemble du système est égale à la plus mauvaise qualité de service**. Nous faisons cette hypothèse dans le but d'offrir la meilleure qualité de service possible dans le pire des cas, au détriment peut-être d'une ou plusieurs commodités qui auraient été "sur-favorisées". Nous voulons aller à l'encontre de ce que nous avons remarqué dans les expérimentations des modèles précédents, à savoir, le fait que quelques commodités sont favorisées par rapport aux autres. Ce phénomène apparaît lorqu'interviennent des commodités avec fortes demandes ou bien il dépend du schéma de résolution utilisé (choix du pivot de l'algorithme, méthode d'utilisation primale, duale ou primale-duale, ...).
Ainsi, la qualité de service globale peut s'écrire :

$$S = QoS = \max_x \min_{k \in K} QoS^k(f^k)$$

Si l'on considère que la qualité de service est inversement proportionnelle au temps de d'acheminement, cela revient à minimiser $\max_{k_\in K} t^k$ (à un coefficient multiplicateur près, multiplicateur différent pour chaque commodité)

On peut remarquer que l'on cherche en quelque sorte le meilleur **diamètre** du réseau **induit** par le système de transport, notamment si l'ensemble des commodités décrit l'ensemble de tous les couples origine/destination. Bien entendu, ce problème est bien plus difficile que le problème du diamètre classique qui lui est polynomial, dans la mesure où le système de transport reste une inconnue.

Pour modéliser cette nouvelle fonction objectif avec le formalisme de la programmation linéaire, il faut introduire une nouvelle variable, appelons-la t, définie sur $\mathbb{R}^+$ et rajouter les K contraintes suivantes :

$$t^k \leq t \quad \text{ou encore} \quad \sum_{e \in E} f_e^k \leq t, \forall k$$

Bien évidemment, ces contraintes sont écrites à un facteur près pour t^k et la fonction objectif se réduit à minimiser t.

Nous verrons l'impact de cette formulation de la qualité de service à la section 2.5. Nous constaterons en particulier son intérêt dans la mesure où certaines commodités ne sont pas favorisées arbitrairement par rapport à d'autres. La fonction objectif est en quelque sorte "égalitaire". Nous verrons aussi que ce problème est expérimentalement plus difficile à résoudre le problème initial.

2.4.4. Obtenir une seule ligne

Comme nous l'avons déjà évoqué, les modèles donnés sont a priori multitournée, c'est-à-dire que la structure de flot de l'objet inconnu, le système de transport, ne permet pas de connaître le nombre de lignes effectivement manipulées. Avec la description de x, on sait que l'on obtient des circuits. Ces circuits peuvent être distincts ou avoir des points de contacts. On peut aussi obtenir des grandes tournées. Pour des raisons pratiques, on peut vouloir identifier chacun de ces circuits et créer plusieurs circuits d'une grande tournée en exploitant les points de contact par exemple.

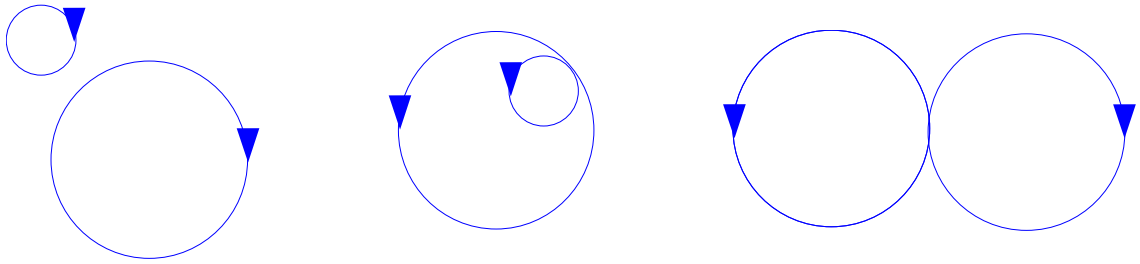


FIG. 2.8 – Exemples de circuits que l'on voudrait identifier

Ce travail peut être effectué par l'organe décisionnel mais on peut aussi vouloir que cette analyse soit réalisée par l'outil d'optimisation lui-même. Quelles sont donc les différentes techniques que l'on peut mettre en œuvre pour n'obtenir qu'une seule et unique tournée ?

Pour être sûr de n'obtenir qu'une seule tournée, on peut s'inspirer des travaux sur le Problème du Voyageur de Commerce, sur le Problème du Voyageur de Commerce Sélectif et sur le Problème de Tournées de Véhicules et introduire ce qui est appelé **des contraintes d'élimination des sous-tours**, c'est-à-dire l'élimination de tous les cycles parasites qui font que nous n'avons pas qu'un seul circuit (on ne peut alors repasser à un point déjà desservi).

Il existe différentes formulations, plus ou moins efficaces suivant la méthode de résolution et qui n'ont pas le même comportement vis-à-vis de la relaxation : l'énumération des sous-tours possibles, l'introduction d'une variable booléenne de passage, l'introduction d'un ordre de passage sur les sommets, l'utilisation de la capacité du véhicule, l'utilisation d'un multiflot véhicule. Intéressons-nous maintenant à quelques unes de ces méthodes et analysons comment les intégrer.

Énumération des sous-tours

Dantzig et al [DFJ54] ont donné une des premières formulations d'exclusion de sous-tours pour le TSP : elle consiste à ajouter ("énumérer") des contraintes qui visent a casser tous les sous-tours possibles.

Si n est le nombre de sommets et y_e une variable qui vaut 1 si l'arc e appartient à la tournée et 0 sinon, on a les contraintes suivantes :

$$\sum_{e\in(U\times U)\cap E} y_e \leq |U|-1, \forall U \subset V, 2 \leq |U| \leq n-2$$

$$\text{équivalentes à} \quad \sum_{u\in U}\sum_{e\in\omega^+(u)} y_e \geq 1, \forall U \subset V, 2 \leq |U| \leq n-2$$

Bien évidemment, cette énumération est exponentielle, de l'ordre de 2^n contraintes d'élimination de sous-tours. De nombreuses façons d'exprimer l'énumération ont été étudiées, mais elles n'ont pas toutes la même qualité [GP99].

On peut facilement intégrer ces contraintes dans notre modèle. Si $MAX_e = 1, \forall e$, il suffit d'intégrer les contraintes sus-citées directement en prenant comme variables des contraintes les variables x_e ($y_e = x_e, \forall e \in A$). Dans le cas contraire, il faut ajouter les variables y_e et les lier aux variables de flot x_e de cette manière :

$$\begin{cases} y_e - x_e \geq 0 \\ (MAX_e + 1)y_e - x_e \geq 0 \end{cases}$$

Ordre de passage

En 1960, *Miller et al* [MTZ60] ont proposé une autre expression qui utilise l'ordre de passage des véhicules. Si on appelle u_i^k l'ordre de passage du conteneur k au sommet i :

$$u_i^k - u_j^k + (n-1)x_{ij}^k \leq n-2 \quad \text{avec } e=(i,j) \in A$$

Cependant, cette formulation, si elle ne comporte que n^2 contraintes, a un bien plus **mauvais comportement vis-à-vis de la relaxation lagrangienne**. Le facteur devant x_e^k peut se transformer en un temps maximal de service si l'on s'intéresse à des fenêtres de temps, ou en capacité maximale des véhicules si l'on considère les demandes [GUE99, p24]. De telles contraintes éliminent naturellement les sous-tours si les fenêtres de temps ou les capacités des véhicules ne sont pas trop grandes. Les contraintes ont été aussi améliorées par **renforcement** [DL91, KLB04].

Flot

Une des meilleures techniques pour les problèmes de type TSP et VRP consiste à manipuler des contraintes de multiflot pour le(s) véhicule(s) [GG78]. On crée un flot $f_{(i,j)}^k$ pour le véhicule k et l'arc (i,j), y_i^k est une variable qui vaut 1 si le véhicule k visite le sommet i et 0 sinon (avec les notations du problème CVRP). Il suffit d'écrire la consommation de flot pour tout sommet différent du dépôt : $\sum_j f_{(i,j)}^k - \sum_j f_{(j,i)}^k = y_i^k$ et au dépôt, on a le flot maximal tel que $\sum_i f_{(0,i)}^k = \sum_i y_i^k$ (où le sommet 0 est le départ de la tournée)

Les contraintes d'élimination de sous-tours deviennent un peu plus compliquées lorsqu'il faut tenir compte d'un point de passage obligé [FEI01, p14], [FT88].

Conclusion

Nous avons donc proposé plusieurs méthodes pour intégrer les contraintes d'élimination de sous-tours dans notre modèle à partir des travaux sur les problèmes du voyageur de commerce et de tournées de véhicules. Toutes les propositions ne sont pas faciles à intégrer et peuvent augmenter la complexité du problème original.

Cependant, il faut noter qu'**expérimentalement**, sortir cette artillerie n'est pas obligatoire. En effet, si la borne sur le coût de x, $p_{\max}$, n'est pas trop grande, nous n'obtenons, **en pratique**, qu'une seule tournée.

2.4.5. Prendre en compte des temps d'attente

Supposons maintenant que l'on veuille prendre en compte les temps d'attente, c'est-à-dire des temps d'accès au système de transport décrit par les conteneurs. Ces temps interviennent naturellement aussi bien dans un réseau où l'on transporte des passagers (connexions, fréquences de passage moins importantes) que pour le transport des biens où il faut souvent considérer des temps de manutention pour la charge et la décharge de ceux-ci.

Pour ce faire, il suffit de modifier le graphe de travail et d'introduire des arcs représentant les temps d'attente lors de la montée dans les conteneurs et des arcs de durée nulle dans l'autre sens (voir Fig 2.10). Pour de plus amples explications sur les modifications à apporter au graphe, le lecteur se rapportera au point suivant où la prise en compte des temps d'attente est une composante essentielle.

2.4.6. Gérer un nombre de lignes fixé

On a vu que le modèle général était à priori multitournée. Nous avons vu aussi comment obtenir une seule tournée au paragraphe 2.4.4. Ce problème est difficile mais dans certains cas peut expérimentalement se traduire par la détermination d'une borne sur la taille du flot conteneur. Si l'on suppose que l'on peut facilement identifier les lignes, on peut chercher à constituer I lignes au maximum.

Hypothèses

Pour ce modèle, on se référera aux hypothèses du modèle simple : déplacement des produits au plus rapide, des conteneurs de capacité infinie. Chaque nouvelle ligne i que l'on propose a un **coût de mise en service** L^i et est associée à la variable binaire y_i qui vaut 1 si la ligne est en service et 0 sinon. On supposera aussi qu'il y a un **délai d'attente moyen** ω_i identique en tout point de la ligne i (Bien entendu, cela ne pose pas de problème de considérer des temps différents pour les points d'accès w_v^i si l'on dispose des données adéquates). Pour chaque ligne obtenue, on ne vérifiera pas que cette ligne n'est bien qu'une seule ligne. On ne considérera plus un coût global ,$p_{\max}$, pour l'ensemble des tournées mais un **coût maximal par tournée** noté $p_{\max}^i$.

Structure du graphe multicouche

Pour modéliser le problème, on a besoin de modifier le graphe de travail. Nous n'allons plus utiliser le graphe $G = (V, E = A \cup \overline{A})$, mais une extension $\tilde{G} = (\tilde{V}, \tilde{E})$ de celui-ci.

On va manipuler un graphe structuré en couches : la couche "lente" et une couche rapide pour chaque ligne possible.

La "couche lente" est définie comme suit : on fait une copie de V que l'on appelle V_0 et une copie de $\overline{A}$ que l'on appelle E_0.

Pour chaque "couche rapide", on copie V_A et A. Ainsi, par abus de notation, on peut poser : $V_i = V_A$ et $E_i = A$.

Il ne reste plus qu'à définir les arcs de transition entre chaque couche ainsi définie. Les lignes ne communiquent pas entre elles, il est obligatoire de passer par la couche "lente". On relie chaque sommet $v \in V_A$ de V_0 au sommet correspondant de la couche i par un arc de coût ω_i (on crée ainsi l'ensemble des arcs T_i^1) et un arc retour de coût nul (appartenant à T_i^0). On appelle $T_i = T_i^0 \cup T_i^1$ l'ensemble de tous ces arcs de transition pour la couche i.

On a alors : $\tilde{V} = \bigcup_{i=0}^{I} V_i$ et $\tilde{E} = E_0 \cup \bigcup_{i=1}^{I} (E_i \cup T_i)$ et les cardinalités associées :

- Il y a n sommets pour la couche lente et Iv pour les couches rapides, d'où $|\tilde{V}| = n + Iv$.
- Il y a $(m - a)$ arcs pour la couche lente, Ia pour les couches rapides et $2Iv$ pour les transitions entre les couches, d'où $|\tilde{E}| = m - a + I(2v + a)$

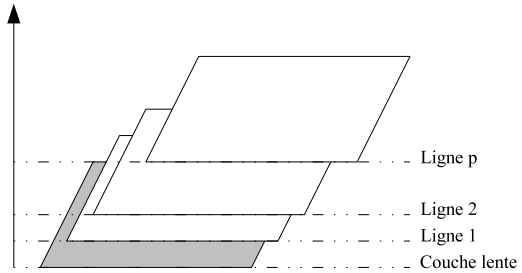


FIG. 2.9 – Modèle multicouche

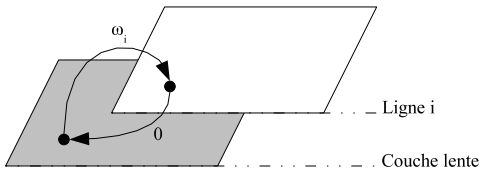


FIG. 2.10 – Transitions entre les couches

Le modèle multicritère est le suivant :

Trouver sur un réseau $\tilde{G} = (\tilde{V}, \tilde{E})$, un flot conteneur $x = (x_e)_{e\in\tilde{A}}$ un multiflot produit $f = (f_e^k)_{e\in\tilde{E}}^{k\in K}$ et un choix de lignes $y = (y_i)_{i\in I}$ tels que

$$\begin{cases} \sum_{k\in K} D^k \sum_{e\in\tilde{E}} f_e^k \\ \sum_{i=1}^{I} L^i y_i \end{cases} \quad \text{soit } \textbf{minimale}$$

sous les contraintes

$$\sum_{e\in E_i} p_e x_e \leq p^i_{\max} \qquad \forall i > 0 \tag{22.1}$$

$$\sum_{e\in\omega^-(u)} x_e - \sum_{e\in\omega^+(u)} x_e = 0 \quad \forall u \in V_i, \forall i > 0 \tag{22.2}$$

$$\sum_{e\in\omega^-(u)} f_e^k - \sum_{e\in\omega^+(u)} f_e^k = b_u^k \qquad \forall u \in \tilde{V}, \forall k \tag{22.3}$$

$$f_e^k \leq x_e \quad \forall e \in \bigcup_{i>0} E_i, \forall k \tag{22.4}$$

$$x_e \leq y_i \quad \forall e \in E_i, \forall i > 0 \tag{22.5}$$

$$0 \leq x_e \leq MAX_e,\ x_e \in \mathbb{N} \qquad \forall e \in \bigcup_{i>0} E_i \tag{22.6}$$

$$f_e^k \in [0,1] \qquad \forall e \in \tilde{E}, \forall k \tag{22.7}$$

$$y_i \in \{0,1\} \qquad \forall i > 0 \tag{22.8}$$

(22.9)

connaissant $\tilde{A} \subset \tilde{E}$, des commodités $(D^k, o^k, d^k)_{k\in K}$, des coûts $(p^i_{\max})_{i\in I}$, $(L^i)_{i\in I}$, $(p_e)_{e\in A}$ et $(c_e)_{e\in E}$ et des capacités $(MAX_e)_{e\in A}$

Pl. 22 : Modèle multicouche

Les contraintes sont en tout point semblables à celles déjà écrites pour les PL précédents. Seules les contraintes (22.5) sont originales : on ne peut se déplacer sur la ligne i que si celle-ci est ouverte. En toute rigueur, il faudrait ajouter des contraintes d'exclusion de sous-tours sur chacune des tournées.

Taille du problème

Variables	
$\mathbb{N}$	$I(a+1)$
$\mathbb{R}$	$I(aK+2vK)+K(m-a)$

Contraintes	
égalités	$Kn + I(a+vK)$
"inégalités"	$K(2m-2a) + I(3Ka+4Kv+3a)$
	$K(2m-2a+n) + I(3Ka+5Kv+4a)$

Le nombre de tournées I a été mis en facteur pour visualiser la complexité spatiale liée à ce paramètre.

Prendre en compte le nombre de changement(s) de lignes

On peut essayer de forcer un nombre minimal de changement(s) de lignes en introduisant dans la fonction objectif le terme : $\min \sum_{e \in T_i} f_e^k, \forall i, \forall k$.

On peut aussi imposer une limite sur ce nombre de changements : on définit pour la commodité k, la variable cht^k qui représente le nombre de changement(s). On a alors l'égalité suivante :

$$\text{cht}^k = \sum_{i=1}^{I} \sum_{e \in T_i} f_e^k - 1$$

On peut alors facilement introduire une contrainte sur le nombre maximal de changements (K nouvelles contraintes dans le programme linéaire).

Remarque

En pratique, dans un contexte d'aide à la décision, on peut enlever les variables y_i en testant le PL pour différentes valeurs de I. Le programme est alors bien plus facile à résoudre et la fonction objectif n'est plus forcément multicritère...

Conclusion

Nous avons présenté un modèle pour prendre en compte un nombre fixe de lignes de transport du flot conteneur. La complexité spatiale de la formulation est telle qui est inenvisageable de résoudre un tel problème avec des outils de résolution comme CPLEX. Nous proposons une modification de la structure du graphe de travail qui est alors "multicouche". Cette notion est primordiale pour la prise en compte de nouveaux paramètres comme les temps d'attente ou de manutention (on pourra se référer aux travaux de simulation présentés dans l'annexe C).

2.4.7. Imposer un point de passage

Dans le modèle général, on laisse le processus d'optimisation déterminer complètement la tournée. Cependant, il est parfois requis de passer en certains points du réseau (dépôt, gare, centre touristique ou commercial, zone de stockage ou de maintenance ...) Si l'on veut qu'une ligne passe obligatoirement par un sommet du graphe donné, il suffit d'imposer une valeur strictement positive lors de l'écriture des contraintes de conservation du flot (x) en ce point.

Si $\tilde{v} \in V$ est un tel point, il faut modifier les contraintes de conservation du flot conteneur (contrainte 20.2 page 38 par exemple) par les suivantes :

$$\sum_{e \in \omega^+(\tilde{v})} x_e = \sum_{e \in \omega^-(\tilde{v})} x_e > 0$$

L'inégalité est bien stricte. Avec le problème mixte original, il suffit de prendre 1 comme valeur minimale. Si l'on travaille avec les contraintes d'intégrité relâchées, il faut une valeur seuil ϵ suffisamment grande pour considérer que le flot passe effectivement en un point donné :

une valeur supérieure au "zéro" de la machine, ou, au contraire, une valeur qui ne laisse place à l'ambiguïté (0.5 par exemple).

Avec de telle contraintes, le flot conteneur, et donc le système de transport, passera obligatoirement par le point choisi.

2.4.8. Changer le modèle de déplacement : le mode "paresseux"

Tous les modèles présentés dans ce chapitre font état d'un déplacement des produits suivant la règle du **déplacement au plus rapide** que l'on peut encore appeler de plus court chemin (PCC). On va tenter dans ce paragraphe de donner des pistes pour modéliser un autre type de déplacement que l'on a surnommé **le mode "paresseux"**. Mais tout d'abord, définissons ce mode de déplacement...

Quand il s'agit de passagers, ceux-ci sont le plus souvent "paresseux", c'est-à-dire que les usagers accèdent au système de transport au plus près de leur origine et en sortent au plus près de leur destination. Ils n'ont que faire si le déplacement dans le véhicule qui les transportent (bus par exemple) fait un trajet un peu plus long et tortueux que le trajet le plus rapide. Ce mode de transport peut aussi trouver une application dans le transport de marchandises. En effet, le transport de biens induit des temps — le plus souvent non négligeables — de chargement et de déchargement. Si l'on prend en compte ces temps additionnels aux temps de transport, on ne peut pas toujours appliquer la règle de déplacement au plus rapide si le déplacement au plus rapide nécessite plusieurs manipulations des marchandises....

Les figures 2.11 et 2.12 montrent la différence de comportement pour les deux modes.

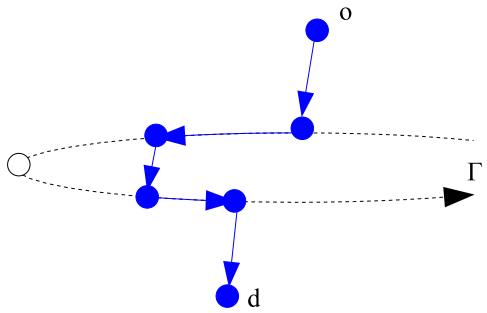


FIG. 2.11 – Déplacement PCC

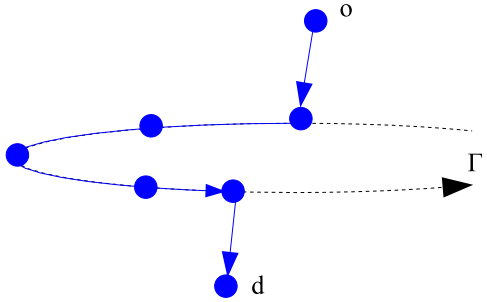


FIG. 2.12 – Déplacement "paresseux"

Supposons qu'un produit doivent être acheminé d'une origine o à une destination d en empruntant une seule ligne de transport Γ avec le mode de déplacement réaliste. On appelle o' le point où le produit est chargé dans le conteneur et d' le point où il est déchargé. Son trajet pourra se décomposer en trois segments : le premier en vitesse lente oo', le deuxième en vitesse rapide $o'd'$ et le dernier à nouveau en vitesse lente $d'd$, comme suit :

$$\overset{\frown}{od} = \overset{p\frown}{oo'} + \overset{\Gamma\frown}{o'd'} + \overset{p\frown}{d'd}$$

Dans cette décomposition du déplacement, les déplacements en vitesse lente $\overset{p\frown}{oo'}$ et $\overset{p\frown}{d'd}$ sont les plus courts possibles et conditionnent les arrêts o' et d'. Bien entendu, on choisira le temps de trajet le plus court si plusieurs trajets sont possibles.

La première proposition de modélisation du déplacement "paresseux" modifie la fonction-objectif qui devient multicritère. La seconde modifie le graphe de réseau afin d'être capable de compter les changements de modalités.

Première proposition

On modifie la fonction-objectif (qui devient multicritère) pour intégrer une minimisation du trajet effectué en vitesse lente.

On ajoutera un des objectifs suivants :

$$\begin{cases} \min \sum_{k\in K, e\in\overline{A}} c_e f_e^k \text{ dans une version agrégée} \\ \text{ou} \\ \left(\min_{e\in\overline{A}}(c_e f_e^k)\right)_{k\in K} \text{ dans une version éclatée} \end{cases}$$

La formulation éclatée est plus réaliste que la formulation agrégée dans la mesure où tous les trajets à pied sont minimisés et que la formulation agrégée peut lisser l'évaluation en compensant certains trajets. Cependant, la formulation agrégée doit être plus facile à prendre en compte en n'ajoutant qu'un seul objectif supplémentaire (la formulation éclatée en rajoute K).

Seconde proposition

On reprend l'idée développée dans le modèle multicouche où interviennent les temps d'attente. Nous cherchons à pénaliser les trajets de PCC. On insère une nouvelle couche entre la couche lente et les couches rapides que l'on appelle la passerelle, comme le montre la figure (2.13). Cette couche supplémentaire représente l'ossature du réseau de transport. C'est le passage obligé pour monter dans un conteneur (arc de transition de coût, le temps d'attente) ou en descendre (arc de coût nul). Cette nouvelle couche et la couche en vitesse lente sont complètement interconnectées et on cherche à minimiser le nombre d'arcs montants (ou descendants). Il n'y a pas d'arc dans cette couche en général, sauf si les arcs représentent une réalité physique comme une passerelle, un tunnel d'accès ou un tapis roulant entre des points proches du réseau.

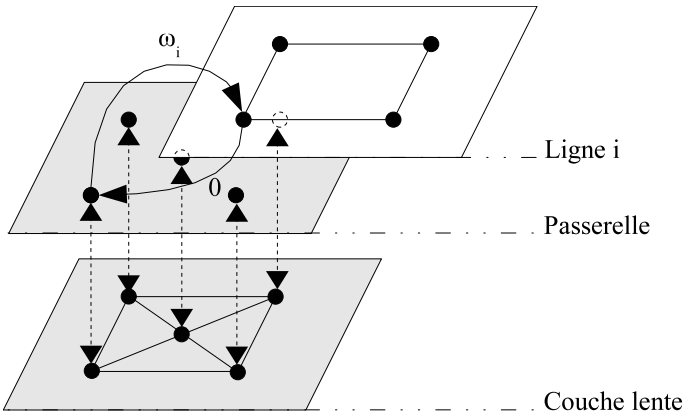


Fig. 2.13 – Modèle multicouche et déplacement paresseux

On peut se passer de cette couche supplémentaire en interconnectant toutes les couches rapides entre elles mais cela augmenterait encore plus la taille du graphe de travail.

On cherche à minimiser soit le nombre d'arcs de transition de la couche lente à la nouvelle couche pour limiter le nombre de connexions, ou alors on le borne.

Conclusion

Nous avons proposé plusieurs pistes pour modéliser un déplacement alternatif que l'on a qualifié de "paresseux". La première fait appel aux techniques de résolution multicritère et la seconde à un changement de structure du graphe de travail.

En pratique, si l'on considère des temps d'attente, des temps de chargement et déchargement non négligeables devant le temps de trajet, le déplacement paresseux s'impose de lui-même. Le modèle prenant compte uniquement ces temps d'attente suffit, il n'est pas nécessaire d'utiliser d'autres artifices de modélisation.

2.4.9. Une ligne "plate"

On veut chercher une ligne non plus sous forme d'un circuit mais sous forme d'une **ligne "plate"**, c'est-à-dire que le trajet aller du conteneur est aussi (en sens inverse) son trajet retour.

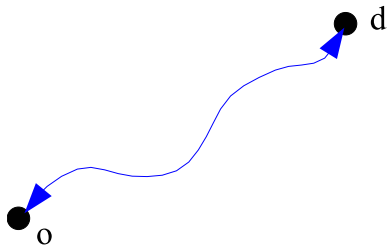


FIG. 2.14 – Ligne "plate" entre o et d

Pour cela, on restreint l'ensemble A à l'ensemble des arcs e qui ont un arc inverse e^{-1} dans A, et on rajoute les contraintes suivantes :

$$x_e = x_{e^{-1}}, \forall e \in A$$

On ne peut pas vérifier l'unicité de la ligne avec les méthodes que l'on a citées dans ce chapitre.

2.5. Expérimentations

Dans cette section, nous allons nous intéresser aux résultats donnés par un solveur de programmes linéaires sur deux modèles exposés dans ce chapitre. Nous nous intéressons tout d'abord au modèle que nous avons qualifié de simple avec des fonctions de satisfaction linéaires par morceaux. Nous comparons sur des instances données les solutions exactes et les solutions où les contraintes d'intégrité du flot conteneur sont relâchées. Nous nous intéressons ensuite à une fonction de satisfaction non séparable (ce que nous avons comparé à la recherche du diamètre induit du graphe). Nous comparons ce modèle et le modèle obtenu en relâchant les contraintes d'intégrité du flot conteneur. Nous comparons ensuite les deux modèles entre eux.

2.5.1. Présentation

Environnement matériel et logiciel

Tous les tests ont été réalisés avec une machine dont la configuration est la suivante :

- Machine : station HP Xeon 2.8Ghz avec 2 Go RAM
- Système : LINUX Gentoo avec le noyau 2.6 compilée pour processeurs i686.
- Solveur : Ilog CPLEX 8.0 (interface *"Callable Library"*)
- Compilateur : GCC/G++ 3.3.2-rc5 avec option d'optimisation -O2

Notons que CPLEX n'est pas un simple solveur de programmes linéaires. Ce logiciel intègre des méthodes très sophistiquées et à la pointe du développement sur la recherche concernant la résolution des programmes linéaires mixtes (Branch&Cut, Points intérieurs, ...). Le logiciel a été paramétré pour utiliser la méthode considérée comme la plus efficace sur le type de problème envisagé : l'approche duale.

Hypothèses

Les réseaux utilisés ont pour support des graphes **triangulés fortement connexes** de différentes tailles (de 10 sommets à 90 sommets) aléatoires pour les plus petits (correspondant aux instances a10e, a10a, a20e et a20a) et le plus grand à l'image du centre-ville de Clermont-Ferrand (cf12 et cf992). Les instances sont décrites en détail à l'annexe B (page 155). Certaines ont peu de commodités, les autres sont "complètes" ($n(n-1)$ commodités). L'ensemble des arcs rapides, A, est une **copie exacte** de l'ensemble des arcs en vitesse lente et ainsi $m = 2a$.

Un **ratio de vitesse uniforme** b_r a été appliqué entre un arc en vitesse lente et un arc en vitesse rapide. Nous avons choisi un ratio de $b_r = 0.33$. Ce ratio correspond par exemple au ratio entre la vitesse moyen d'un piéton et celle d'un bus en milieu urbain.

Les coûts de la tournée (p_e) ont été exprimés en terme de temps de parcours. Ainsi, nous avons pris , $p_e = c_e, \forall e \in A$. La borne de coût sur le système de transport s'identifie donc à un temps de parcours maximal.

Pour chaque commodité k, les temps de références ont été choisis comme tels : t_0^k est le temps de trajet en vitesse lente sur le plus court chemin entre o^k et d^k et t_1^k est le temps entre ces deux mêmes sommets en vitesse rapide. Le déplacement des produits se fait suivant la règle du **routage au plus rapide** (le chemin d'un produit est un plus court chemin entre l'origine et la destination, PCC induit par le système de transport).

En général, nous n'avons autorisé qu'**un seul passage** d'un conteneur **par arc** et ainsi $MAX_e = 1, \forall e \in E$ (nous avons gardé cette contrainte par la suite dans la mesure où cela, expérimentalement, ne change pratiquement rien).

Nous avons limité le temps de calcul des différentes instances à 2h.

Note sur les graphiques présentés

Sauf mention contraire, les graphes représentent la satisfaction obtenue (sans unité) en fonction d'une borne sur la somme des tailles des tournées. Les graphes étant générés aléatoirement ou à l'échelle d'une carte réelle, les longueurs présentées n'ont qu'un intérêt tout relatif. On pourra se reporter à l'annexe B (page 155) pour l'ordre de grandeur de ces distances. Nous donnons dans la même unité la longueur du plus petit arc et du plus grand arc, ainsi que celle du diamètre.

La valeur de la satisfaction optimale dépend naturellement du nombre de commodités considérées et de leur pondération ($S_{\max} = \sum_k D^k$).

2.5.2. Premier modèle

Le premier modèle que nous avons testé est le modèle simple PL20 (à la page 38) avec la qualité de service la plus simple possible (donnée à la section 2.4.1). Appelons le modèle M_1. Pour ce modèle, on minimise la fonction objectif suivante : $Z_1 = \sum_{k \in K} D^k \frac{t^k}{t_0^k}$ qui correspond à la satisfaction S_1 :

$$S_1 = \sum_{k \in K} \frac{D^k}{1 - b_r} \left(1 - \frac{t^k}{t_0^k}\right) = \frac{1}{1 - b_r} \left[\sum_{k \in K} D^k - Z_1\right]$$

Le programme linéaire associé au modèle présente un nombre de variables difficiles (entières) de l'ordre du nombre d'arcs rapides sur le graphe. On peut donc s'attendre à être très vite limité au niveau de la taille du graphe. Nous avons donc cherché aussi à comparer les résultats du modèle brut avec le modèle où les contraintes d'intégrité sur le flot conteneur x ont été relâchées. On appelle M_{1r} le problème dérivé de M_1 où ces contraintes d'intégrité ont été relâchées.

La réplication d'une instance est paramétrée par le coût maximal sur le système $p_{\max}$. On note $p = \sum_{e \in A} p_e x_e$ le coût effectif de la solution obtenue par la résolution. On ajoute l'indice $_r$ pour toutes les valeurs des modèles relâchés.

Cette comparaison se fera, entre autres, à travers deux indicateurs $\Delta Z = \left|\frac{Z_r - Z}{Z_r}\right|$ et $\Delta p = \left|\frac{p_r - p}{p_r}\right|$, c'est-à-dire la **différence relative** au niveau de la **fonction objectif** et la différence relative au niveau des **coûts** sur le système.

Instance a10e [10 sommets, peu de commodités]

La figure 2.15 montre la satisfaction obtenue pour M_1 (S_1) et celle obtenue avec le modèle relâché M_{1r} (S_{1r}) en fonction de la borne $p_{\max}$. La figure 2.16 exprime toujours en fonction de $p_{\max}$ la différence relative entre la satisfaction S_1 et S_{1r}.

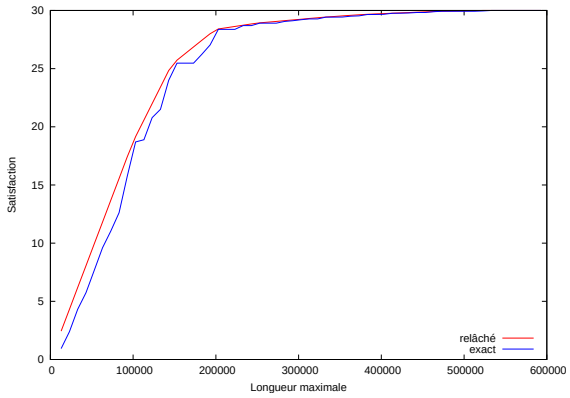


FIG. 2.15 – S_1 et S_{1r} pour a10e

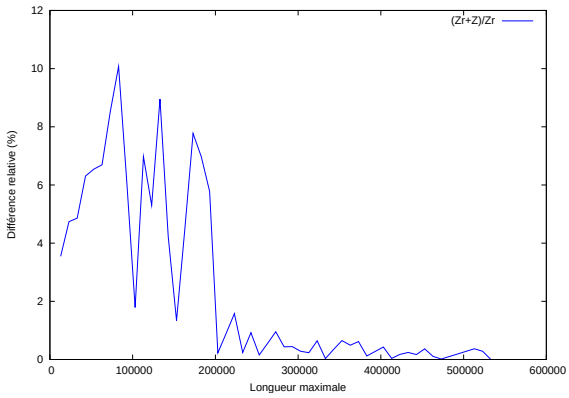


FIG. 2.16 – Comparaison entre M_1 et sa relaxation pour a10e

nombre de réplications	59	
à l'optimum	59	
itérations	min	177
	max	11105
	moy	2145
comparaison ΔZ	max	10.08 %
	moy	2.10 %
comparaison Δp	max	10.52 %
	moy	2.71 %

TAB. 2.2 – Statistiques - $M1$ - a10e

Le tableau ci-contre, 2.2, donne quelques statistiques concernant la résolution avec le réseau a10e : le nombre de réplications traitées, le nombre d'instance résolues à l'optimum. Le nombre minimal d'itérations effectuées par CPLEX (c'est-à-dire le nombre d'instances de simplexe résolues), le plus grand nombre et la moyenne. Nous donnons ensuite les différents indicateurs ΔZ et Δp avec le plus grand écart et l'écart moyen.

L'échelle d'observation se fait avec différents $p_{\max}$. Pour trouver le plus grand $p_{\max}$ possible, on a tout d'abord résolu le problème linéaire en relâchant la contrainte et en évaluant ce paramètre. Pour un coût supérieur à ce coût limite, aucun changement ne se produit sur la solution obtenue.

On compare le modèle M_1 avec sa relaxation M_{1r} en évaluant les satisfactions correspondantes à la figure 2.15. On peut voir, figure 2.16, que l'écart est d'autant plus grand que $p_{\max}$ est — relativement — petit. En effet, quand la borne augmente, tout se passe comme si la contrainte sur le flot conteneur était elle-aussi relâchée : la taille n'est plus importante alors une tournée est composée de trajets directs entre les couples origine-destination et de nombreux détours "à vide". La borne théorique est donc atteinte pour les grandes valeurs de $p_{\max}$.
La contrainte sur le coût du système est relativement serrée (Δp moyen est de l'ordre de 3%).

Instance a10a [10 sommets, commodités point à point]

La figure 2.17 montre la satisfaction obtenue pour M_1 (S_1) et celle obtenue avec le modèle relâché M_{1r} (S_{1r}) en fonction de la borne $p_{\max}$. Le tableau 2.3 comporte exactement le même type de données que le tableau précédent.

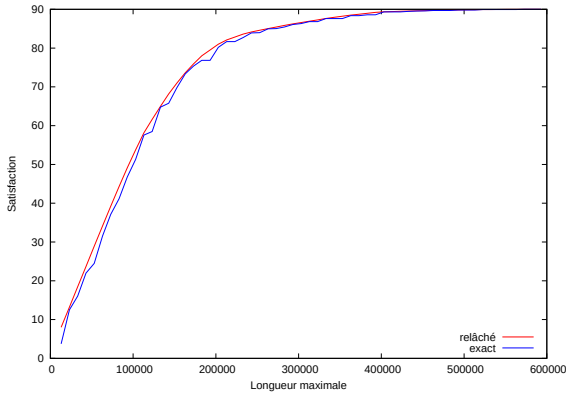


FIG. 2.17 – S_1 et S_{1r} pour a10a

nombre de réplications	59	
à l'optimum	59	
itérations	min	418
	max	14438
	moy	2926
comparaison ΔZ	max	4.88 %
	moy	1.10 %
comparaison Δp	max	10.87 %
	moy	2.64 %

TAB. 2.3 – Statistiques - M_1 - a10a

On peut faire les mêmes remarques que pour l'instance précédente. Nous pouvons juste ajouter que la différence entre le modèle et sa relaxation est plus faible. Cela est certainement du à la densité de commodités (le réseau est complet à ce niveau) sur cette instance. Quelles que soient les lignes générées pendant le processus de résolution, elles concernent toujours un certain nombre de commodités qui, si elles ne sont pas routées tout à fait au plus rapide, sont tout de même de point à point.

Instance a20e [20 sommets, peu de commodités]

La figure et le tableau représentent le même type de données que précédemment.

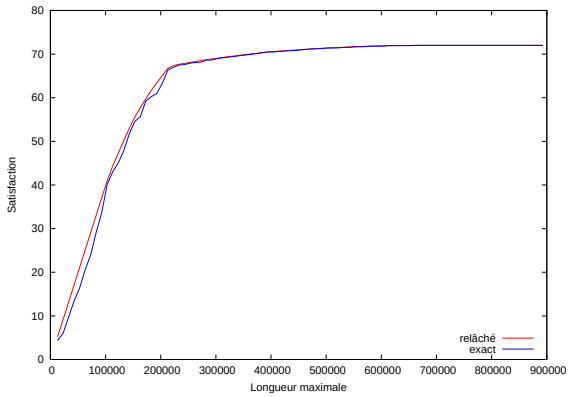


FIG. 2.18 – S_1 et S_{1r} pour a20e

nombre de réplications	89	
à l'optimum	89	
itérations	min	793
	max	237839
	moy	20063
comparaison ΔZ	max	6.07 %
	moy	0.92 %
comparaison Δp	max	5.47 %
	moy	1.87 %

TAB. 2.4 – Statistiques - M_1 - a20e

On observe bien les mêmes tendances que pour les instances à 10 sommets. A noter que toutes les instances sont encore résolues à l'optimum.

Instance a20a [20 sommets, commodités point à point]

La figure et le tableau représentent le même type de données que précédemment.

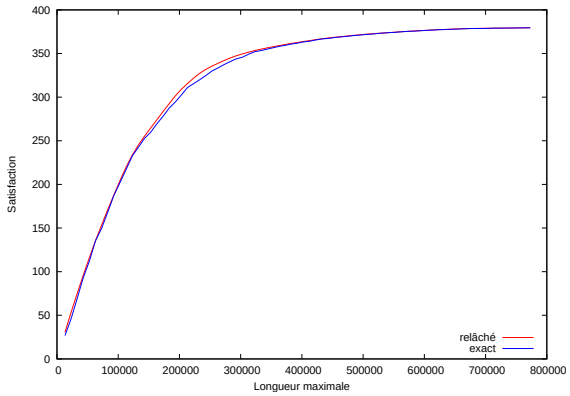


FIG. 2.19 – S_1 et S_{1r} pour a20a

nombre de réplications	77	
à l'optimum	77	
itérations	min	3731
	max	221689
	moy	41140
comparaison ΔZ	max	3.08 %
	moy	0.70 %
comparaison Δp	max	8.74 %
	moy	0.61 %

TAB. 2.5 – Statistiques - M_1 - a20a

Instance cf12 [92 sommets, 12 commodités]

La figure et le tableau représentent le même type de données que précédemment. Au vu de la taille du réseau sous-jacent, la limite sur le temps de calcul a été repoussée à 6h.

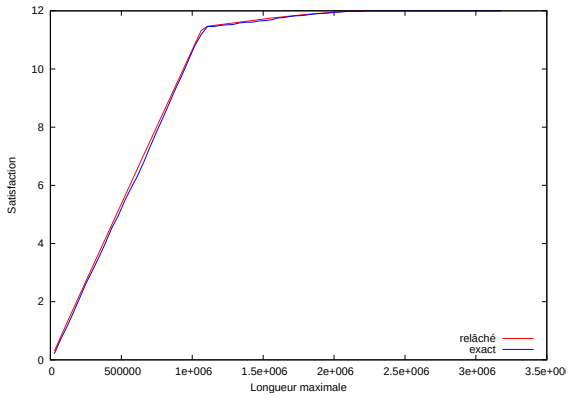


FIG. 2.20 – S_1 et S_{1r} pour cf12

nombre de réplications	71	
à l'optimum	71	
itérations	min	613
	max	56132
	moy	11736
comparaison ΔZ	max	2.31 %
	moy	0.56 %
comparaison Δp	max	14.64 %
	moy	1.42 %

TAB. 2.6 – Statistiques - M_1 - cf12

On peut remarquer sur cette instance que le modèle relâché est encore plus proche du modèle exact.

Instance cf992 [92 sommets, 992 commodités]

Cette instance comporte 992 commodités. Nous avons lancé la résolution du modèle relâché sur 154 instances et 34 n'ont pas pu être résolues (avec une limite sur le temps de calcul de 6h). La résolution du modèle entier mixte n'est même plus envisageable.
La courbe à la figure 2.21 montre que l'on n'a pas de borne valable pour des petites valeurs de $p_{\max}$

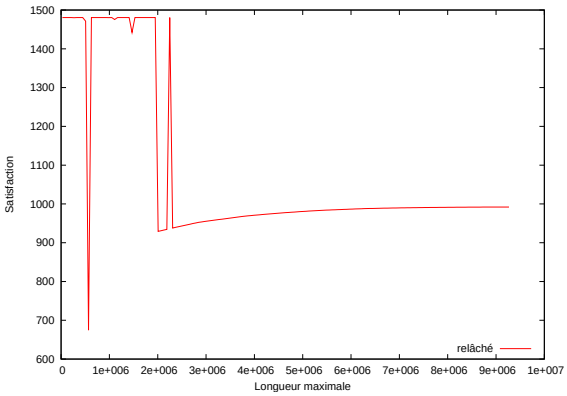


FIG. 2.21 – S_{1r} pour cf992

Commentaires

La **contrainte sur le coût du flot conteneur est primordiale**. Dans le cas contraire, le comportement du modèle est l'acheminement point à point et finalement, le système de transport a un très mauvais rapport qualité de service / coût. De nombreuses portions deviennent inutiles. On peut donc suggérer que notre **modèle est peu adapté pour les grandes valeurs de $p_{\max}$**. Il aurait peut-être fallu intégrer un **contrôle de la taille** du système dans la fonction objectif. Cependant, ajouter un terme en ce coût dans la fonction objectif soulève d'autres problèmes : ceux de l'optimisation multicritère. En effet, les deux **critères** sont totalement **divergents** — le critère de qualité de service tend à l'augmentation de la taille du système jusqu'au transport point à point et s'oppose donc à une minimisation de facto — et une linéarisation de la fonction objectif peut mener à l'oubli de nombreuses solutions non dominées.

Pour de **faibles coûts** sur le système de transport, on obtient le plus souvent qu'**une seule ligne** (au sens où on l'a définie à la section 2.4.4), ce qui ne justifie pas l'emploi des différentes techniques citées dans ce paragraphe qui alourdissent la résolution.

Les contraintes sur le nombre de véhicules par arc sont parfois importantes. Dans le cas où l'on prend un coût $p_{\max}$ relativement élevé et des conteneurs de capacité infinie, le processus de résolution peut amener à trouver des lignes "doubles" (ou plus, la ligne est parcourue plusieurs fois par des véhicules) qui en fait ne sont pas meilleures que des lignes simples (parcourue une seule fois).

Ce modèle s'intéresse à la qualité de service agrégée générée par le système de transport. On peut donc dire que globalement le routage des produits est satisfaisant en moyenne. Maintenant, on peut s'intéresser à la qualité des différents routages, un par un. Il faut donc, au passage, calculer explicitement le temps de parcours des différentes commodités t^k. On remarque alors que pour optimiser la fonction objectif, le processus de résolution favorise telle ou telle commodité au plus rapide et met de côté les autres commodités. A demandes égales, on ne peut donc pas prévoir à priori quelles seront les commodités qui seront favorisées et celles qui ne le seront pas.

Pour compenser un tel défaut, on peut, dans un cadre d'un logiciel d'aide à la décision, aider le programme de résolution en imposant des bornes sur les temps d'acheminement et imposer une qualité minimale sur telle ou telle commodité.

Les premiers tests de exposés dans cette thèse ont d'abors été réalisés avec ILOG CPLEX 6.0. Entre les deux versions utilisées, les gains de résolution — temps de calcul diminué, fermeture du gap — ont été impressionnants. En effet, certaines instances à 10 nœuds n'étaient pas toutes résolues alors que maintenant, ces petites instances sont résolues à l'optimalité.

2.5.3. Deuxième modèle

Nous venons de voir un des principaux défauts du premier modèle : le caractère agrégé de la fonction de satisfaction. On peut donc se poser la question de la pertinence du modèle présenté à la section 2.4.3 (page 45) où l'on cherche à optimiser la plus mauvaise qualité de service. Nous appelons M_2 ce deuxième modèle et M_{2r} le modèle où les contraintes d'intégrité sur le flot conteneur ont été relâchées.

On garde les mêmes hypothèses de résolution que pour le premier modèle. A des fins comparatives, nous allons minimiser la fonction objectif suivante :

$$Z_2 = \max_{k \in K} \left\{ D^k \frac{t^k}{t_0^k} \right\}$$

On a donc $Z_2 \leq Z_1 = \sum_{k \in K} D^k \frac{t^k}{t_0^k} \leq K.Z_2$.

Nous allons pouvoir comparer le modèle M_1 avec le modèle M_2 en calculant leurs satisfactions respectives S_1 et S_2 en tant que somme des satisfaction des différentes commodités (alors S_2 est la satisfaction du modèle M_2 calculée comme si c'était la satisfaction du modèle M_1)

Instance a10e [10 sommets, peu de commodités]

La figure 2.22 montre l'évolution de la fonction objectif obtenue pour M_2 (Z_2) et celle obtenue avec le modèle relâché M_{2r} (Z_{2r}) en fonction de la borne $p_{\max}$. La figure 2.23 fait de même avec la satisfaction calculée sur le mode de M_1. La figure 2.24 compare les satisfactions obtenues avec M_1 et avec M_2.

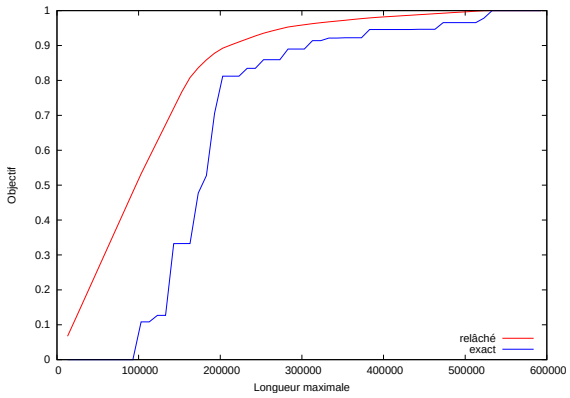


FIG. 2.22 – Z_2 et Z_{2r} pour a10e

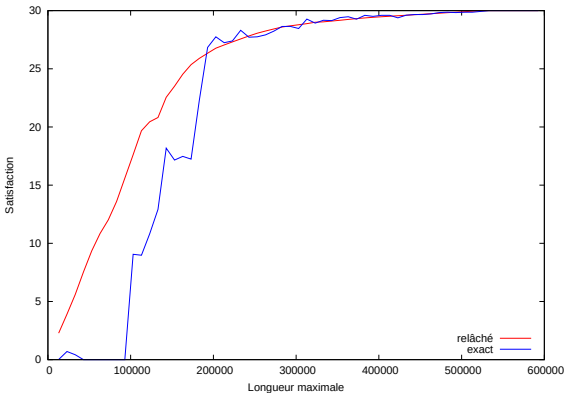


FIG. 2.23 – S_2 et S_{2r} pour a10e

On peut tout d'abord noter que le modèle ne permet pas d'avoir une qualité de service non nulle (au sens de la pire qualité de service du système) pour de petites valeurs de $p_{\max}$. La courbe s'anime lorsque toutes les qualités de service sont non nulles. Pour plus de cohérence, nous avons éliminé ces valeurs du tableau de statistiques : la fonction objectif n'est pas assez discriminante et nous n'avons pas assez d'informations.
On peut aussi remarquer que la valeur de la fonction objectif relâchée est bien plus éloignée de la valeur que l'on cherche à optimiser. L'estimation avec le modèle relâché est donc vraiment plus imprécise. On voit même que l'on ne peut plus être sûr d'un encadrement en ce qui concerne l'évaluation de la satisfaction (S_2 oscille autour de S_{2r}).

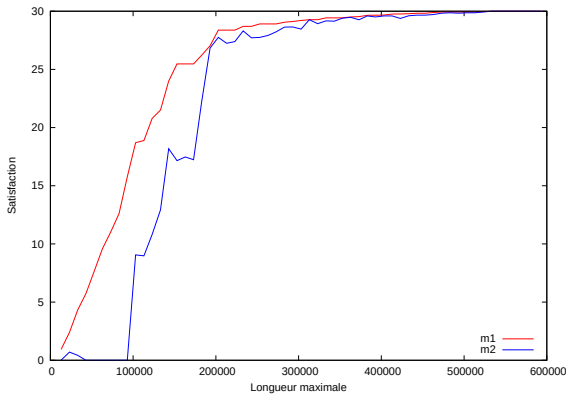


FIG. 2.24 – S_1 et S_2 pour a10e

nombre de réplications	59	
à l'optimum	59	
itérations	min	152
	max	299026
	moy	34385
comparaison ΔZ	max	69.49 %
	moy	17.29 %
comparaison Δp	max	8.77 %
	moy	2.54 %
comparaison ΔS	max	42.63 %
	moy	6.57 %
comparaison ΔM	max	30.11 %
	moy	6.24 %

TAB. 2.7 – Statistiques - M_2 - a10e

Dans cette dernière figure, nous comparons la satisfaction (au sens du premier modèle, c'est-à-dire qualité de service agrégée) entre les deux modèles. On peut donc dire que si on a réussi à obtenir une meilleure "moins mauvaise" qualité de service individuelle, cela se passe au détriment de la qualité de service globale qui est meilleure si l'on se contente de n'optimiser que quelques déplacements.

On retrouve ce genre de résultat lorsque l'on oblige un produit à emprunter une ligne de transport, et cela, même si le trajet est avant optimisation plus long que celui de plus court chemin en vitesse lente.

Aux données déjà calculées pour le modèle M_1, on ajoute **deux indices de comparaison** : on compare la **satisfaction** donnée par M_2 avec celle donnée par M_{2r} (ΔS) et la satisfaction donnée par M_1 avec celle donnée par M_2 (ΔM).
Ce tableau ne fait que confirmer ce que les graphes laissaient supposer : le modèle M_2 et sa relaxation sont très éloignés l'un de l'autre. La satisfaction obtenue par M_2 est de loin moins bonne que celle obtenue par M_1.

Instance a10a [10 sommets, commodités point à point]

La figure 2.25 montre l'évolution de la fonction objectif obtenue pour M_2 (Z_2) et celle obtenue avec le modèle relâché M_{2r} (Z_{2r}) en fonction de la borne $p_{\max}$ sur l'instance a10a. La figure 2.26 fait de même avec la satisfaction calculée sur le mode de M_1. La figure 2.27 compare les satisfactions obtenues avec M_1 et avec M_2.

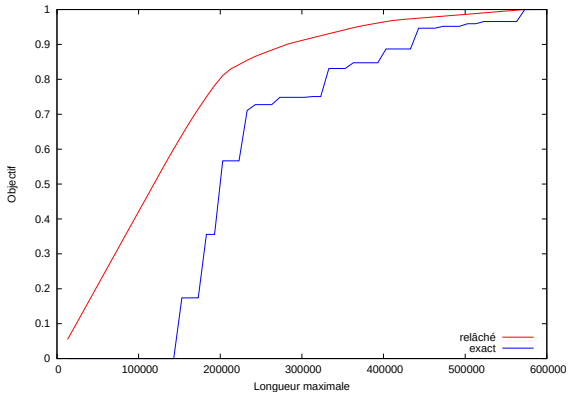


FIG. 2.25 – Z_2 et Z_{2r} pour a10a

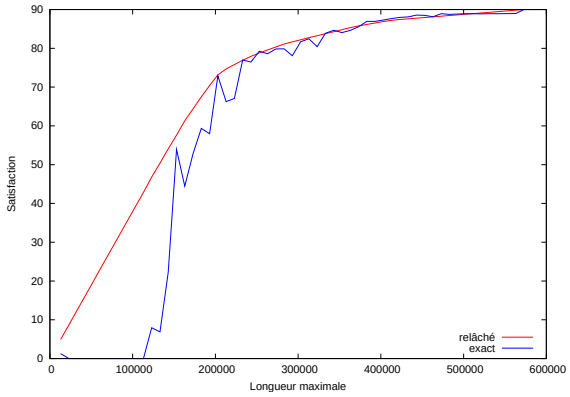


FIG. 2.26 – S_2 et S_{2r} pour a10a

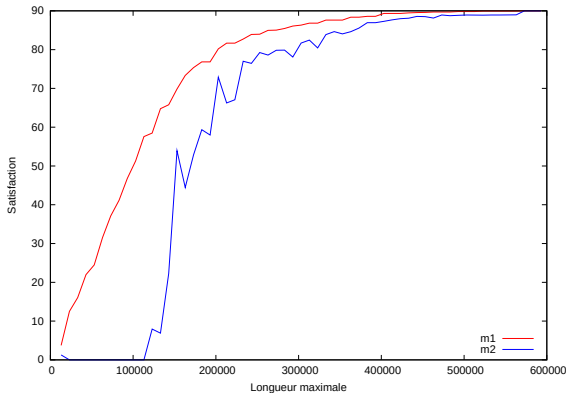


FIG. 2.27 – S_1 et S_2 pour a10a

nombre de réplications	59	
à l'optimum	59	
itérations	min	341
	max	720238
	moy	38716
comparaison ΔZ	max	69.69 %
	moy	26.62 %
comparaison Δp	max	77.18 %
	moy	7.22 %
comparaison ΔS	max	51.86 %
	moy	5.98 %
comparaison ΔM	max	83.21 %
	moy	13.82 %

TAB. 2.8 – Statistiques - M_2 - a10a

Les résultats sont en tous points similaires au cas précédent. Le modèle M_2 a un comportement bien différent de sa relaxation. On peut noter une explosion du nombre d'itérations due simplement à l'élargissement de l'ensemble des commodités.

Instance a20e [20 sommets, peu de commodités]

La figure 2.28 montre l'évolution de la fonction objectif obtenue pour M_2 (Z_2) et celle obtenue avec le modèle relâché M_{2r} (Z_{2r}) en fonction de la borne $p_{\max}$ sur l'instance a20a. La figure 2.29 fait de même avec la satisfaction calculée sur le mode de M_1. La figure 2.30 compare les satisfactions obtenues avec M_1 et avec M_2.

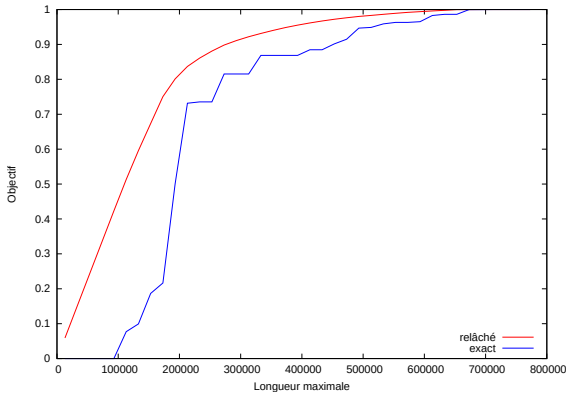


FIG. 2.28 – Z_2 et Z_{2r} pour a20e

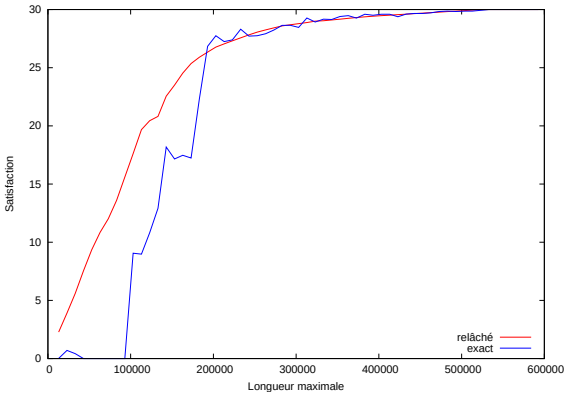


FIG. 2.29 – S_2 et S_{2r} pour a20e

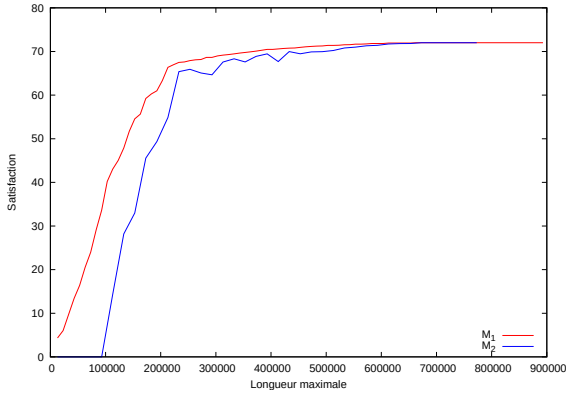


FIG. 2.30 – S_1 et S_2 pour a20e

nombre de réplications	89	
à l'optimum	58	
itérations	min	2514
	max	5056490
	moy	2025953
comparaison ΔZ	max	43.50 %
	moy	5.92%
comparaison Δp	max	3.88%
	moy	0.59%
comparaison ΔS	max	16.75%
	moy	1.38%
comparaison ΔM	max	20.03%
	moy	2.44%

TAB. 2.9 – Statistiques - M_2 - a20e

Bien entendu n'interviennent pas dans le tableau de statistiques les instances dont la résolution n'est pas optimale. 65% des instances ont été résolues à l'optimalité.

Instance a20a [20 sommets, commodités point à point]

La figure 2.31 montre l'évolution de la fonction objectif obtenue pour M_2 (Z_2) et celle obtenue avec le modèle relâché M_{2r} (Z_{2r}) en fonction de la borne $p_{\max}$ sur l'instance a20a. La figure 2.32 fait de même avec la satisfaction calculée sur le mode de M_1. La figure 2.33 compare les satisfactions obtenues avec M_1 et avec M_2.

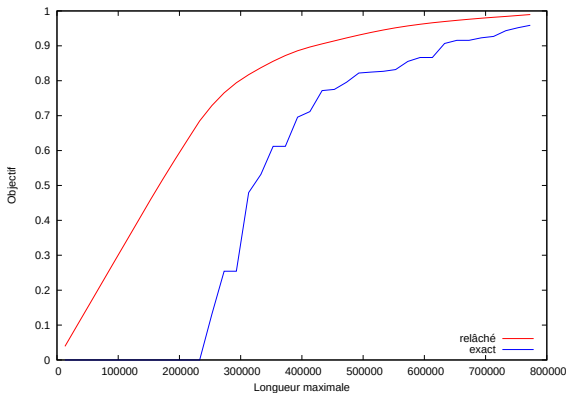


FIG. 2.31 – Z_2 et Z_{2r} pour a20a

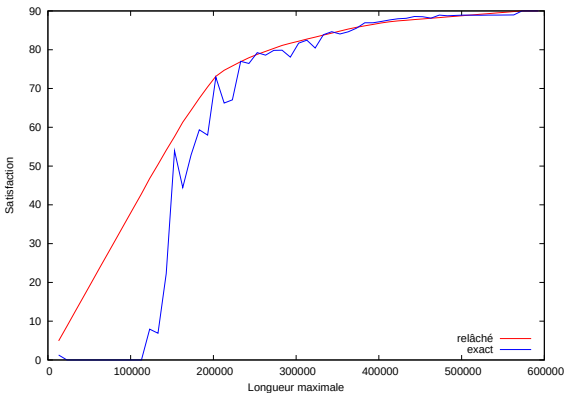


FIG. 2.32 – S_2 et S_{2r} pour a20a

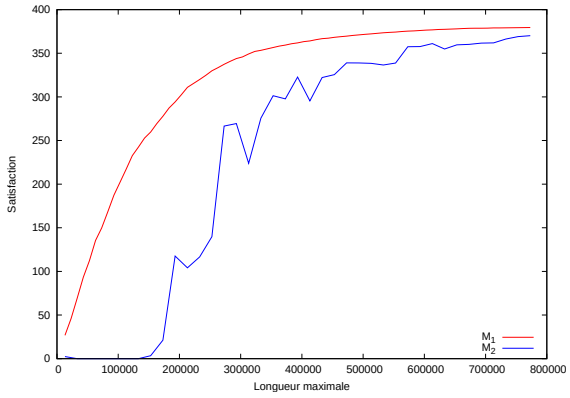


FIG. 2.33 – S_1 et S_2 pour a20a

nombre de réplications	77	
à l'optimum	43	
itérations	min	13413
	max	1008018
	moy	449405
comparaison ΔZ	max	22.28 %
	moy	13.04%
comparaison Δp	max	1.77%
	moy	0.85%
comparaison ΔS	max	11.10%
	moy	5.36%
comparaison ΔM	max	16,02 %
	moy	9.30%

TAB. 2.10 – Statistiques - M_2 - a20a

Seulement 56% des instances ont été résolues à l'optimalité. Le nombre de commodités a bien perturbé la résolution des problèmes.

Les solutions dont l'optimalité n'a pas été obtenue n'interviennent pas dans le tableau des statistiques mais elles sont tout de même représentées sur les courbes. Si l'on fait abstraction de ces dernières, on constate tout de même que la satisfaction S_1 est très supérieure à la satisfaction S_2.

Instances cf12 et cf 992

Le modèle M_2 est bien plus difficile à résoudre et comme la relaxation du modèle ne donne pas une bonne approximation, nous ne donnons pas de résultats.

Commentaires

Le modèle M_2 est un modèle relativement lourd à mettre en œuvre : la complexité pratique est telle qu'il est déjà difficile d'obtenir des solutions mêmes sur des petites instances.

Se servir du modèle relâché M_{2r} à la place du modèle original n'est pas d'un grand secours dans la mesure où les résultats qu'ils renvoient peuvent être très différents. L'estimation par M_{2r} est dans de nombreux cas trop grossière.

2.5.4. Conclusion

Nous avons donc illustré le comportement de deux modèles issus de notre modèle général présenté dans ce chapitre. On a pu remarquer que le solveur de programmes linéaires rencontrait très vite ses limites et qu'il n'était plus possible d'obtenir des solutions exactes.

Cette limitation qui apparaît déjà sur des modèles aussi simples rend illusoires la mise en œuvres de certaines modélisations exposées dans ce chapitre, surtout lorsqu'elles augmentent le nombre de variables entières. Cependant, certaines modélisations peuvent être vérifiées avec d'autres méthodes que la programmation linéaire mixte. Nous verrons ultérieurement comment la prise en compte de temps d'attente permet de valider le comportement "paresseux" des produits (c'est-à-dire en évitant les changements impromptus de lignes).

Malgré ces bémols, nous avons pu constater — notamment en ce qui concerne M_1, c'est-à-dire le modèle qui agrège les différentes fonctions de satisfaction — que la solution donnée par le problème relâché était une bonne indication sur la solution du problème non relâché. Ainsi, l'objectif principal de ce chapitre est atteint : nous disposons de modélisation pour les réseaux de mobilités avec demande élastique et nous disposons d'indicateurs, même grossiers, sur la qualité des solutions considérées. Ces résultats vont donc nous permettre d'évaluer et d'étalonner les différentes métaheuristiques que l'on a adaptées à cette famille de problèmes.

Chapitre 3

Résolution approchée pour les problèmes de synthèse de réseaux de mobilité à demande élastique

Sommaire

Au chapitre 2, nous avons expliqué comment modéliser le problème de transport que constitue le problème de synthèse de réseaux de mobilité avec demande élastique. Les expérimentations numériques, avec un solveur de programmation entière mixte, ont montré sur des modèles simples formalisés en termes de programmes mixtes, qu'il était difficile d'obtenir des solutions de qualité, notamment avec des réseaux de grande taille ou avec un grand nombre de commodités.

Dans ce chapitre, nous nous tournons donc vers la résolution heuristique, approchée, pour obtenir des bonnes solutions sur ces instances qui posent problème. Nous proposons deux adaptations de métaheuristiques classiques, *GRASP* et *Tabou* en l'occurrence, pour résoudre un tel problème. Nous avons utilisé pour cela une structure particulière appelée la géodésique.

Nous allons tout d'abord décrire cette structure, puis nous rappelons les principes de chacune des deux métaheuristiques. Nous proposons ensuite deux manières d'étendre la définition de la géodésique soit pour obtenir une solution plus rapidement, soit pour couvrir un espace des solutions encore plus large. Nous terminerons par quelques expérimentations numériques.

3.1. Une description particulière de tournée : la géodésique

Notre problème est centré sur la recherche d'une structure topologique qui accélère le déplacement sur le réseau. Dans l'optique d'une application aux transports, on cherche un ensemble de lignes que l'on repère le plus souvent par des circuits, encore appelés tournées, sur le réseau.

Dans ce type de problème, les tournées ont une **forme relativement simple**, **lisse** dans le sens où elles ne présentent généralement pas de nombreux repliements, boucles ou encore croisements sur elles-même. Dans cette section, nous allons définir une structure décrivant de manière très simple un circuit, qui semble présenter, à priori, ce type de comportement tout en offrant des facilités de manipulation. Nous avons baptisé cette structure une **géodésique** et nous avons utilisé cette structure dans notre implémentation des différentes métaheuristiques considérées.

3.1.1. Définitions

On appelle **chemin géodésique** un plus court chemin (PCC) entre deux sommets d'un graphe. Soit P un chemin géodésique, on note $origine(P)$ (respectivement $destination(P)$) le premier (respectivement dernier) sommet de P.

On appelle **Q-géodésique** tout ensemble ordonné de Q (avec $Q \geq 2$) chemins géodésiques

$\{P_0, \cdots, P_{Q-1}\}$ tels que :

$$\begin{array}{lll} & origine(P_i) & = destination(P_{(i+Q-1)[Q]}) \quad \forall i \in \{0,..,Q-1\} \\ \text{ou encore} & destination(P_i) & = origine(P_{(i+1)[Q]}) \quad \forall i \in \{0,..,Q-1\} \end{array}$$

où $[Q]$ dénote le *modulo* Q.

Parler de Q-géodésique pour $Q \leq 2$ n'a évidemment aucun sens. La figure (3.1) montre un exemple de 3-géodésique (avec trois chemins géodésiques notés P_0, P_1 et P_2). Par extension, on appelle **géodésique**, toute Q-géodésique dont le nombre de chemins géodésiques n'est pas spécifié.

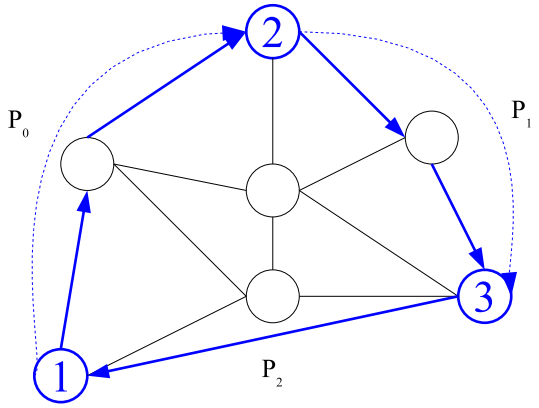


FIG. 3.1 – Exemple de géodésique

On appelle **point de contrôle** de la géodésique toute extrémité d'un chemin géodésique d'une géodésique. Une géodésique a autant de points de contrôle qu'elle a de chemins géodésiques. La géodésique de la figure (3.1) a trois points de contrôle : (1), (2) et (3). On appelle encore **ordre** d'une géodésique le nombre de points de contrôle/de chemins géodésiques. Le modèle UML de la géodésique est donné en annexe par la figure (B.2 page 159).

Notons que l'on ne peut décrire une Q-géodésique complètement par la donnée de ses Q points de contrôle. En effet, on ne peut garantir l'unicité d'un PCC entre deux points d'un graphe.

La géodésique est une caractérisation de **circuit**. En effet, par définition, toute géodésique est un circuit et réciproquement, tout circuit C à P sommets peut être considéré, trivialement, comme une P-géodésique. Trouver l'ordre minimal d'une géodésique correspondante à un circuit C est un problème combinatoire (c'est-à-dire trouver un entier Q minimal tel que C puisse être décrit comme une Q-géodésique), nous ne nous sommes pas attachés à savoir si c'était un problème facile (voir la section 3.1.6).

On dira que deux géodésiques sont **équivalentes** si les deux géodésiques décrivent exactement la même suite ordonnée d'arcs à une rotation près, c'est-à-dire si elles décrivent le même circuit.

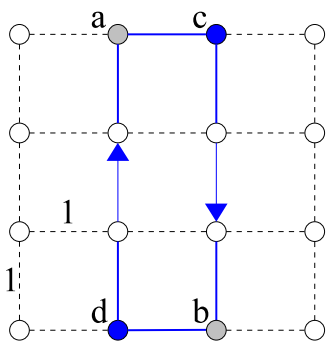


FIG. 3.2 – Exemple de deux géodésiques équivalentes

L'exemple simple donné à la figure (3.2), sur un graphe de type grille, représente un circuit. Ce circuit sert de support à deux 2-géodésiques définies par des chemins géodésiques différents et donc des points de contrôle différents (a) et (b) pour la première, (c) et (d) pour la seconde. Ces deux géodésiques sont donc équivalentes. Remarquons aussi que le nombre de points de contrôle n'est pas un facteur de discrimination entre deux géodésiques : deux géodésiques peuvent avoir un nombre de points de contrôle différents et être équivalentes. Il faut donc réellement comparer les supports des deux géodésiques pour savoir si elles sont équivalentes ou pas (ce qui n'est pas en soi un problème très difficile, mais l'implémentation doit être soigneuse pour ne pas perdre inutilement de temps. Suivant le type de graphe, on peut déjà accélérer le test en comparant la longueur des géodésiques, ou le nombre d'arcs des géodésiques).

3.1.2. Propriétés

La longueur d'une Q-géodésique (c'est-à-dire la longueur du circuit décrit par cette géodésique) ne peut excéder le produit du nombre de points de contrôle, Q, avec le diamètre Δ du graphe (i.e. la plus grande distance de plus court chemin entre deux points du graphe, comme on l'a appelé à la section 2.4.3) :

$$\text{Taille}_{\max} \leq Q.\Delta$$

Affiner cette borne est difficile car cela revient à résoudre un problème de Voyageur de Commerce. En effet, il faut résoudre un problème de voyageur de commerce sélectif pour tout choix de Q sommets sur les n possibles du graphe et en prendre le maximum.

De la même manière, il semble relativement intuitif que l'ordre d'une géodésique conditionne sa forme. Avec un ordre relativement faible, elle sera régulière sans repliement intempestif (par analogie avec le problème de déterminer une courbe passant par un certain nombre de points donnés).

On dira qu'une géodésique est non **dégénérée** lorsque tous les points de contrôles sont distincts.

3.1.3. Modélisation linéaire de la géodésique

Au chapitre 2, nous avons proposé un certain nombre de modèles linéaire pour les problème de réseaux de mobilité avec demande élastique. Ces modèles recherchent comme objet principal un flot véhicule qui décrit, entre autres, un ensemble de tournées de conteneurs circulant sur la partie rapide du réseau.

Nous proposons une modélisation linéaire de la géodésique pour permettre d'**évaluer l'impact du choix d'une telle structure** pour la recherche d'une tournée maximisant la qualité de service du système de transport proposé. A titre d'exemple, on injecte la modélisation linéaire dans le problème de transport avec demande élastique simple (présenté à la section 2.4.1 page 41).

Modélisation

On va modéliser une Q-géodésique. On suppose que la géodésique n'est pas dégénérée, c'est-à-dire que tous les points de contrôle sont distincts.

On considère un graphe $G = (V, E)$ et un sous-ensemble d'arcs $A \subset E$. On note V_A le sous-ensemble de sommets qui sont extrémité d'un arc de A. Soit u_v^q une variable binaire qui vaut 1 si le sommet $v \in V_A$ est le $q^{\text{ième}}$ point de contrôle de la géodésique et 0 sinon.

Soit x_e^q une autre variable binaire : elle est positive si l'arc $e \in A$ est utilisé dans le plus court chemin associé au $q^{\text{ième}}$ point de contrôle et 0 sinon.

Cette variable est à priori entière mais on peut, en pratique, relâcher la contrainte d'intégrité. En effet, ces contraintes ne seront pas entières uniquement dans le cas où il existe plusieurs chemins équivalents. Hors, cela n'a pas d'influence, ni sur la position des points de contrôle dans le réseau, ni sur l'évaluation de la fonction-objectif.

Un seul sommet du sous-graphe (V_A, A) peut être le $q^{\text{ième}}$ point de contrôle, alors on a les contraintes suivantes :

$$\sum_{v \in V_A} u_v^q = 1, \quad \forall q \in \{1, , Q\}$$

Les autres contraintes sont relatives à l'écriture de la conservation de flot en chacun des sommets :

$$\sum_{e \in (\omega^+(v) \cap A)} x_e^q - \sum_{e \in (\omega^-(v) \cap A)} x_e^q = u_v^{q[Q]+1} - u_v^q, \quad \forall v \in V_A, \quad \forall q \in \{1, , Q\}$$

où $[Q]$ représente le *modulo* Q.

Les plus courts chemins sont calculés si l'on minimise les quantités suivantes : $\sum_{e \in A} c_e x_e^q, \forall q \in \{1, , Q\}$.

Exemple d'intégration dans un modèle linéaire

Voici le modèle linéaire du problème de transport avec demande élastique en imposant l'utilisation d'une géodésique comme tournée réalisable :

Trouver sur un réseau $G = (V, E)$, des variables binaires $x = (x_e^q)_{e\in A}^{q\in\{1,,Q\}}$ et un multiflot produit $f = (f_e^k)_{e\in E}^{k\in K}$ tels que $\begin{cases} \sum_{k\in K} D^k.\Phi^k(\sum c_e t_e^k) \\ -\sum_{e\in A} c_e x_e^q, \forall q \in \{1,,Q\} \end{cases}$ soit **maximale** sous les contraintes :

$$\sum_{e\in A}\left(p_e \sum_p x_e^q\right) \leq p_{\max} \tag{23.1}$$

$$\sum_{e\in(\omega^+(v)\cap A)} x_e^q - \sum_{e\in(\omega^-(v)\cap A)} x_e^q = u_v^{q[Q]+1} - u_v^q \qquad \forall v \in V_A, \forall q \tag{23.2}$$

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \tag{23.3}$$

$$f_e^k \leq x_e^q \qquad \forall e \in A, \forall k, \forall q \tag{23.4}$$

$$\sum_{v\in V_A} u_v^q = 1 \qquad \forall q \tag{23.5}$$

$$u_v^q \in \{0,1\} \qquad \forall v \in V_A, \forall q \tag{23.6}$$

$$x_e^q \in [0,1] \qquad \forall e \in A, \forall q \tag{23.7}$$

$$f_e^k \in [0,1] \qquad \forall e \in E, \forall k \tag{23.8}$$

connaissant $A \subset E$, des commodités $(D^k, o^k, d^k)_{k\in K}$,des coûts $p_{\max}$, $(p_e)_{e\in A}$ et $(c_e)_{e\in E}$

Pl. 23 : Exemple d'intégration de la modélisation linéaire d'une géodésique

Toutes les contraintes ont déjà été expliquées pour le Pl 20. Seule la contrainte de coût sur la géodésique est un tout petit peu plus difficile à exprimer. La fonction-objectif est multicritère. Pour donner le modèle à résoudre à un solveur, il faut déterminer un jeu de paramètres pour minimiser ou maximiser une combinaison linéaire des différents objectifs. Déterminer une telle combinaison linéaire doit être possible dans la mesure où les deux objectifs ne sont pas toujours antagonistes : en effet, plus courts sont les trajets, meilleure est la satisfaction. La taille de ce PL est très respectable (particulièrement en ce qui concerne le nombre, critique, de variables entières). Il reste malheureusement très difficile à résoudre même sur de très petites instances.

3.1.4. Voisinages pour heuristiques

Les métaheuristiques que nous avons implémentées sont classées dans la famille des heuristiques de **"recherche locale"**, c'est-à-dire que l'on explore l'espace des solutions (sous-

entendu réalisables) en passant d'une solution à une autre.

Déterminer le **voisinage** d'une solution, c'est déterminer toutes les solutions "voisines", voisines au sens où elles sont obtenues par l'application d'un **opérateur** (dit de voisinage) sur cette solution. On note généralement $V(x)$ le voisinage d'une solution x. Pour de plus amples renseignements sur les notions sous-jacentes aux métaheuristiques et aux types de voisinages, on pourra se référer à la thèse de Christophe Duhamel [DUH01, chapitre 1].

Nous avons dit précédemment que la connaissance des points de contrôle d'une géodésique n'était pas suffisante pour caractériser cette géodésique, nous jouons sur ces "degrés de liberté" pour définir des géodésiques voisines. Ainsi, nous avons identifié cinq types de voisinage :

- le **changement de chemin géodésique** entre deux points de contrôle. A la figure (3.3), on remplace le chemin P_1 entre les points de contrôle (1) et (2) par le chemin P_2. P_1 et P_2 sont des plus courts chemins entre (1) et (2) d'évaluation (en général, de longueur) identique. Bien évidemment, ce changement n'est possible que s'il y a plusieurs plus courts chemins entre ces deux points.

- le **déplacement d'un point de contrôle**. Ce type de mouvement peut modifier radicalement le chemin parcouru entre les points de contrôle adjacents à celui qui vient d'être déplacé. À la figure (3.4), le point de contrôle (3') a été remplacé par (3).

- l'**insertion** ou la **suppression d'un point de contrôle**. Ces modifications structurelles permettent de supprimer par exemple un point de contrôle redondant (lorsque celui-ci se trouve déjà sur un chemin de PCC) ou au contraire permet d'acquérir plus de souplesse sur la forme de la géodésique, comme on peut le voir à la figure (3.5).

- le **réordonnancement** (changement de l'ordre de parcours) des points de contrôle. A la figure (3.6), les points de contrôle (2) et (3) ont été échangés. Ce mouvement définit en quelque sorte un mouvement 2-opt* [PR95] sur la géodésique.

- une **redéfinition complète**, aléatoire ou contrôlée, de la géodésique. Ce type de mouvement, même s'il n'est pas à proprement parler, un opérateur de voisinage, peut permettre de sortir d'un extremum local [DUH01].

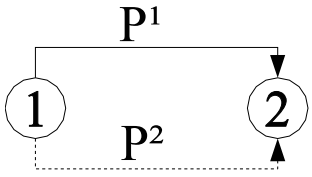


FIG. 3.3 – Changement de PCC

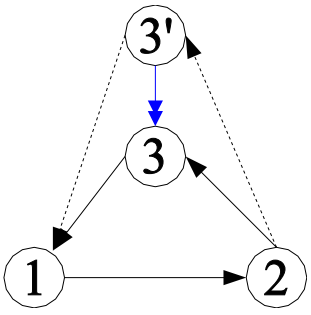


FIG. 3.4 – Déplacement du point (3)

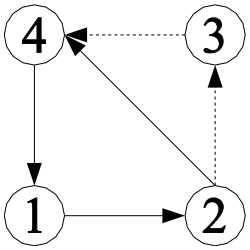


FIG. 3.5 – Insertion/Suppression du point (3)

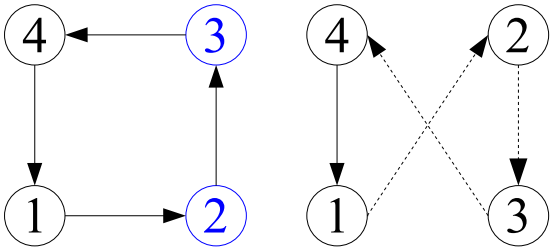


FIG. 3.6 – Échange de deux points

Les voisinages sont intéressants si le nombre de points de contrôle est largement inférieur au nombre de sommets effectivement sur la géodésique. Dans le cas contraire, les mouvements sont très limités (déplacement des points de contrôle).

Avec les différents voisinages, on constate que les opérations sur la géodésique se résument à une manipulation (insertion ou suppression) d'arcs, la seule contrainte étant de conserver la propriété de circuit. On peut donc se poser la question de savoir si ce type de mouvement est **similaire au λ-opt** défini par Lin [LIN65, LK73]. Dans ce type d'échange, λ est un paramètre fixe : on enlève et on remet exactement λ arcs dans la solution. Dans le cas présent, on enlève un certain nombre d'arcs et on en rajoute un certain nombre. Il n'y a pas de contrôle possible sur ces nombres ni aucune relation entre eux.

Tous les voisinages que nous venons de présenter s'appliquent naturellement sur une seule géodésique. Nous verrons au paragraphe 3.6.2 les voisinages propres à la gestion du multi-tournée.

3.1.5. Extension de la définition de la géodésique

Pour les voisinages, on utilise déjà les degrés de liberté offerts lorsque l'on manipule une géodésique avant tout par ses points de contrôle. On va étendre la notion de chemin entre deux points de contrôle. Dans la définition initiale, celle donnée au début de ce chapitre, on énumére tout simplement tous les plus courts chemins entre deux points de contrôle. On souhaite désormais **élargir la famille de chemins possibles** et autoriser des chemins dont l'**évaluation est moins bonne**.

On pourra retenir les chemins entre deux points de contrôle suivant **différents critères** : on pourra par exemple ne retenir que les chemins qui sont **indépendants** (en précisant le type d'indépendance, arcs ou nœuds, et le degré d'indépendance). On pourra aussi ordonner les chemins suivants leur **évaluation** : on ne retenait que les plus courts chemins entre deux points, on peut élargir la liste aux K plus courts chemins (là encore, nous avons le choix : les algorithmes usuels donne un chemin pour chaque valeur, on pourra énumérer tous les chemins pour une valeur donnée) ou encore à tout chemin quelconque. Ces différents chemins pourront être stockés en mémoire, ou bien alors calculés à la volée (utiliser un algorithme incrémental de K PCC, par exemple). Nous discutons du problème des K plus courts chemins à l'annexe A.7.3 page 149.

3.1.6. D'un circuit quelconque à une géodésique

La géodésique est par définition un circuit : si, au cours d'un traitement, on a besoin d'utiliser cette propriété, cela ne pose pas de problème majeur dans la mesure où il suffit de parcourir tous les chemins géodésiques.

Maintenant, on peut se demander si, connaissant un circuit quelconque donné, on peut le considérer comme une géodésique. Cela peut être utile à de nombreux titres. En effet, de nombreux procédures de résolution dans la littérature considèrent que l'objet tournée est un simple circuit et proposent des manipulations spécifiques à la structure de circuit. Si l'on est amené à utiliser de tels algorithmes, on peut avoir à **retourner à la structure de géodésique** par un procédé pas trop trivial.

D'un autre côté, nous avons déjà posé le **problème de l'équivalence** de deux géodésiques, c'est-à-dire pouvoir déterminer si deux géodésiques, à priori différentes, décrivent le même circuit support. Une bonne caractérisation d'un circuit en une géodésique pourrait peut être apporter une **réponse élégante** à ce type de problème (dans le cas qui nous intéresse, cela pourrait être particulièrement utile pour déduire et stocker des informations pour la diversification des heuristiques et pour le processus mémoriel de la métaheuristique tabou).

Nous n'avons pas réfléchi à un algorithme permettant de passer d'un circuit quelconque à une géodésique d'ordre minimal, le problème est **encore ouvert**. Cependant, il est immédiat d'écrire un algorithme de type glouton, très rapide, pour faire une conversion avec un ordre qui, s'il n'est pas minimal, ne doit pas non plus être aussi élevé que le nombre de points dans le circuit. Cet algorithme pour être efficace suppose tout de même que l'on connaisse tous les plus courts chemins entre tous les points du circuit.

Son principe est très simple : on choisit un point du circuit au hasard, c'est le premier point de contrôle. On choisit ensuite un point du circuit, dans le sens de parcours du circuit, qui constitue le plus grand PCC avec le point choisi précédemment. Le chemin ainsi déterminé constitue le premier chemin géodésique de la géodésique. On itère le processus, comme le montre l'algorithme (2), jusqu'à ce que l'on ait couvert complètement le circuit initial.

On suppose que l'on dispose d'un circuit $\mathcal{C}$, d'un point quelconque de ce cicuit, nommons-le Q_0 et d'une fonction $suivant(P)$ qui donne le point suivant du point P dans le circuit $\mathcal{C}$. On va déterminer un ensemble de points $Q_{i\geq 0}$ correspondant à un ensemble de points de contrôle d'une géodésique dont le support est $\mathcal{C}$.

Algorithme 2 : Circuit $\longrightarrow$ Géodésique

Entrée : un circuit $\mathcal{C} = (Q_0, \text{suivant})$
Retourne : des points de contrôle $Q_{i\geq 0}$
$i \leftarrow 0$, $Q \leftarrow Q_0$
tant que $(suivant(Q) \neq Q_0)$ **faire**
  tant que $(suivant(Q) \neq Q_0)$ **ET** ($\overline{Q_i, suivant(Q)}$ est un PCC) **faire**
    $Q \leftarrow suivant(Q)$
  fin tant que
  si $suivant(Q) \neq Q_0$ **alors**
    $Q_i \leftarrow Q$, $i \leftarrow i + 1$
  fin si
fin tant que

3.2. Métaheuristique GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) est apparue dans les années 90 (voir [FR89, FR95]). Elle apporte une réponse simple et élégante au problème soulevé par l'exploration globale de l'espace des solutions pour les procédures de recherche locale. La phase de recherche locale est encapsulée dans un mécanisme plus général de génération contrôlée de solutions (voir [DUH01]).

L'heuristique GRASP alterne les phases de recherche locale (intensification) et les phases de diversification, comme le montre l'algorithme simplifié (5).

3.2.1. Recherche locale

Lors de la recherche locale, l'algorithme permet de prendre quelques libertés vis-à-vis de l'algorithme de descente simple. Il n'est pas requis de complètement réaliser la phase de descente : on peut prendre toute solution qui améliore la solution courante. Nous pouvons procéder de plusieurs façons. Dans notre implémentation, on se contente de fixer un nombre maximal d'itérations pour cette phase de recherche locale. Si la descente est rapide, ce critère n'intervient pas. Si, au contraire, la descente est plutôt lente, la phase de descente est arrêtée lorsque le nombre d'itérations est atteint et on relance la recherche dans une autre région de l'espace des solutions. On note $V(x)$ le voisinage d'une solution réalisable x.

3.2.2. Diversification

La **diversification** est le nom que l'on donne à l'opération qui consiste à générer une solution durant le processus de résolution dans le but avoué d'explorer une zone différente de l'espace des solutions. On essaie, dans la mesure du possible, de **tirer profit de certaines informations relevées** pendant les itérations précédentes pour réellement toucher une région différente. L'idée sous-jacente est de couvrir la plus grande zone possible de l'espace des solutions de manière cohérente. Au final, l'alternance des phases de diversification et des phases de recherche locale (intensification) doit permettre de trouver les meilleures solutions possibles.

L'objet que nous manipulons est la géodésique décrite à la section précédente. La géodésique caractérise un circuit qui peut être vu, soit comme une suite ordonnée d'arcs, soit comme une suite ordonnée de nœuds. Il faut trouver des informations susceptibles de donner une bonne diversification.

... orientée arcs

On applique le principe suivant : on pose un **compteur d'utilisation** pour chaque arc de A, ce compteur est incrémenté à chaque fois qu'un arc est utilisé dans une solution courante de l'algorithme. La nouvelle tournée sera générée avec des arcs dont l'utilité est moindre par rapport aux autres.

On doit se contenter d'un algorithme simple. On ne cherche pas à trouver la meilleure tournée au sens de non-utilisation des arcs et avec contrainte de taille, car ce problème est un problème de tournées sur les arcs bien connu : le problème du postier rural (ou *Rural Postman Problem* [GUE99]) ou encore un problème de tournées sur les arcs profitables [FEI01]. Le premier problème peut être résolu en temps polynomial mais il faut encore déterminer une géodésique représentant le circuit.

... orientée sommets

Au vu des difficultés citées précédemment et de la description de l'objet cherché — la description de la géodésique privilégie les sommets que sont les points de contrôle — il semble plus judicieux de mettre un compteur d'utilisation sur les sommets.

Là encore, plusieurs alternatives s'offrent à nous. On peut compter l'utilisation de tous les points de la géodésique, ou uniquement des points de contrôle.

Expérimentalement, nous avons constaté que mesurer l'utilité de tous les sommets ne donne pas de résultat intéressant. La suite des géodésiques générées de cette manière était cyclique avec une période relativement courte. Pour briser ce cycle, il est nécessaire perturber l'indice d'utilité en introduisant de l'aléatoire.

Algorithmes

Nous donnons deux algorithmes de principe pour diversifier, le premier est dit **multistart** (algorithme 3), c'est-à-dire que l'on n'utilise pas les informations des itérations précédentes et le second est à mémoire sur les nœuds (algorithme 4). Ils permettent tous deux d'obtenir une Q-géodésique (où $2 \leq Q \leq n$) ne dépassant pas une certaine taille. La démarche de construction est la même : on cherche tout d'abord une 2-géodésique réalisable (qui vérifie la contrainte de taille maximale) et on insère, tant que c'est possible (tant que la contrainte de taille n'est pas violée) un nouveau point de contrôle en dernière position de la liste des points de contrôle déjà définis. **L'ordre effectif peut donc être inférieur à l'ordre demandé**. La question qui est de savoir si un sommet du graphe est insérable, est relative au maintien de la réalisabilité de la solution courante : un sommet n'est inséré en tant que point de contrôle que si la géodésique résultant reste réalisable.

Dans l'algorithme 3, on maintient deux sous-ensembles de sommets du graphe. Le premier ensemble, s_1, est composé des points de contrôle N_i déjà déterminés. Le deuxième ensemble, s_2, est réinitialisé à chaque nouvelle recherche d'un point de contrôle, il contient tous les sommets déjà explorés en tant que points de contrôle (avérés ou testés mais refusés). L'algorithme s'arrête lorsque le nombre de points de contrôle requis Q est atteint ou lorsqu'il n'est plus possible d'insérer de nouveaux points de contrôle (l'insertion de n'importe quel nouveau sommet viole la contrainte de réalisabilité). L'initialisation de l'algorithme se fait en choisissant au hasard deux sommets de graphe qui servent de premiers points de contrôle pour la géodésique sous réserve que la contrainte sur la longueur de la géodésique est satisfaite. L'algorithme retourne une erreur si cette phase d'initialisation n'est pas réalisable.

Algorithme 3 : Diversification multistart

Entrée : Q, un nombre de points de contrôle ≥ 2
Retourne : Une géodésique de points de controle $s1 = \{N_i\}_{i \leq Q}$
trouver une 2-géodésique réalisable (de points de contrôle N_1 et N_2)
$s_1 \leftarrow \{N_1, N_2\}$, $q \leftarrow 2$
$Stop_1 \leftarrow (q = Q)$
tant que non $Stop_1$ **faire**
  $Stop_2 \leftarrow$ **faux**, $s_2 \leftarrow s1$
  tant que non $Stop_2$ **faire**
    choisir $N_q \notin s_2$ au hasard
    si N_q insérable **alors**
      insérer N_q dans la géodésique courante
      $s_1 \leftarrow s_1 \cup \{N_q\}$, $q \leftarrow q + 1$, $Stop_2 \leftarrow$ **vrai**
      si $q = Q$ **alors** $Stop_1 \leftarrow$ **vrai**
    sinon
      $s_2 \leftarrow s_2 \cup \{N_q\}$
      si $|s_2| = n$ **alors** $Stop_1 \leftarrow$ **vrai**
    fin si
    $Stop_2 \leftarrow Stop_2$ **OU** $Stop_1$
  fin tant que
fin tant que

En ce qui concerne le deuxième algorithme de diversification, l'algorithme (4), on suppose que l'on dispose d'un outil permettant de classer dynamiquement (au cours de la résolution) des nœuds, une mesure d'utilité par exemple. Pratiquement, les nœuds sont rangés suivant cette mesure dans une file à priorité notée PQ.

Algorithme 4 : Diversification avec mémoire sur les nœuds

Entrée : Q, un nombre de points de contrôle ≥ 2
Retourne : Une géodésique de points de controle $\{N_i\}_{i \leq Q}$
ordonner les nœuds dans une file à priorité PQ
$N_1 \leftarrow$ tête de PQ, défiler tête de PQ
défiler PQ jusqu'à trouver N_2 tel que l'on obtienne une 2-géodésique réalisable
$q \leftarrow 2$
tant que $(q < Q)$ **ET** (PQ non vide) **faire**
  $N_{q+1} \leftarrow$ tête de PQ, défiler tête PQ
  tant que N_{q+1} n'est pas insérable **ET** PQ non vide **faire**
    $N_{q+1} \leftarrow$ tête de PQ, défiler tête PQ
  fin tant que
  si N_{q+1} est insérable **alors**
    insérer N_{q+1} dans la géodésique courante
    $q \leftarrow q + 1$
  fin si
fin tant que

Remarquons que nos algorithmes de construction de géodésique ne permettent pas qu'un sommet soit utilisé plusieurs fois comme point de contrôle d'une même géodésique.
Comme nous l'avons déjà écrit, l'insertion d'un nouveau point de contrôle se fait en fin de liste des points de contrôle déjà déterminés. Nous aurions peut-être pu essayer de trouver la meilleure place d'insertion (comme pour les heuristiques avec recherche du meilleur coût d'insertion [LS98]). Notre méthode est peut-être un peu trop respective, mais elle prohibe l'insertion de points trop éloignés et dans la mesure où l'on obtient tout de même une géodésique réalisable, ce ne nous semble pas important.

3.2.3. Algorithme

Nous donnons maintenant l'algorithme de principe, simplifié, de la méthode GRASP. La recherche locale est une descente classique, d'une solution à une autre solution voisine améliorante. Lorsque l'on a atteint un minimum local, on génére une nouvelle solution de manière gloutonne en utilisant l'information synthétisée lors des descentes précédentes.

On note $V(x)$ le voisinage d'une solution x, c'est-à-dire l'ensemble des solutions voisines au sens ouù elles sont obtenues par un opérateur de voisinage. Les différents opérateurs monotournée sont présentés à la section (3.1.4).

Paramètres :

- un nombre maximal d'itérations $\text{iter}_{\max}$
- un nombre maximal de diversifications $\text{div}_{\max}$
- un nombre maximal d'itérations dans la recherche locale $\text{liter}_{\max}$
- une fonction objectif f à minimiser

Lexique :

- x est la solution courante
- x^* est la meilleure solution trouvée
- *iter* est le nombre d'itérations de l'algorithme
- d est le nombre de diversifications déjà réalisées
- *liter* est le nombre d'itérations de la recherche locale

L'algorithme s'arrête lorsque le nombre maximal d'itérations ($\text{iter}_{\max}$) ou le nombre maximal de diversifications a été atteint ($\text{div}_{\max}$). On renvoie alors la meilleure solution trouvée. La recherche locale peut être arrêtée lorsque le nombre d'itérations est trop grand ($\text{liter}_{\max}$).

Un paramètre dans cet algorithme est implicite, il s'agit de la taille maximale de la géodésique. Il est dissimulé aussi bien pour la diversification, que lors de la recherche d'un voisin admissible. La solution voisine doit être améliorante mais réalisable avant tout.

L'algorithme (simplifié) de GRASP est le suivant :

Algorithme 5 : Algorithme de principe de GRASP

$d \leftarrow 1$, $iter \leftarrow 1$, $x^* \leftarrow \emptyset$
tant que $(d \leq \text{div}_{\max})$**ET**$(iter \leq \text{iter}_{\max})$ **faire**
  $\rightsquigarrow$ diversification
  construire une nouvelle géodésique x
  incrémenter $iter$, incrémenter d

  $\rightsquigarrow$ phase de recherche locale
  générer $V'(x) = \{x' \in V(x)/f(x') < f(x)\}$, $liter \leftarrow 0$
  tant que $(V'(x) \neq \emptyset)$ **ET** $(liter \leq \text{liter}_{\max})$ **ET** $(iter \leq \text{iter}_{\max})$ **faire**
    sélectionner $x' \in V'(x)$
    incrémenter $liter$, incrémenter $iter$
    valider le mouvement $x \leftarrow x'$
    générer $V'(x) = \{x' \in V(x)/f(x') < f(x)\}$
  fin tant que

  $\rightsquigarrow$ mise à jour de la valeur record
  si $f(x) < f(x^*)$ **alors**
    $x^* \leftarrow x$
  fin si
fin tant que

3.2.4. Expérimentations

Présentation des tests

Nous reprenons les instances de test décrites an annexe page 155 que nous avons déjà utilisées dans la mise en œuvre des modèles au chapitre 2 (section 2.5 page 55).

Nous utilisons, au choix, deux modes de transport pour les piétons : le **mode PCC** et le **mode réaliste**. L'évaluation d'une géodésique avec l'un de ces modes est l'opération la plus fréquemment appelée dans les algorithmes. Il est donc nécessaire qu'elle soit la plus rapide possible. On trouvera dans [ZN98] une comparaison entre les différents algorithmes de plus courts chemins pour des réseaux réels (les auteurs remarquent que les réseaux générés aléatoirement ont souvent un ratio arcs/nœuds bien plus élevé que les réseaux réels). On peut remarquer aussi, notamment en ce qui concerne le mode PCC, que le coût des arcs varie peu lors de la recherche locale : il n'est pas nécessaire de réévaluer entièrement les solutions. Ce problème a été déjà largement traité sous le nom du problème de plus court chemin dynamique, encore appelé incrémental (on pourra se référer par exemple à [RR96, DFMN00, BRT03] et à [DEI04] pour une comparaison de différentes méthodes). Ces algorithmes, incrémentaux, ont une complexité au pire égale aux algorithmes de PCC usuels.

Nous allons comparer les résultats obtenus par la résolution par Ilog CPLEX, quand nous disposons de ces informations, et ceux obtenus par la méthode GRASP que nous avons implémentée. Nous avons testé **deux variantes de GRASP** : la première avec une diversification aléatoire, c'est-à-dire sans extraire d'informations lors des phases de recherche locale, et la

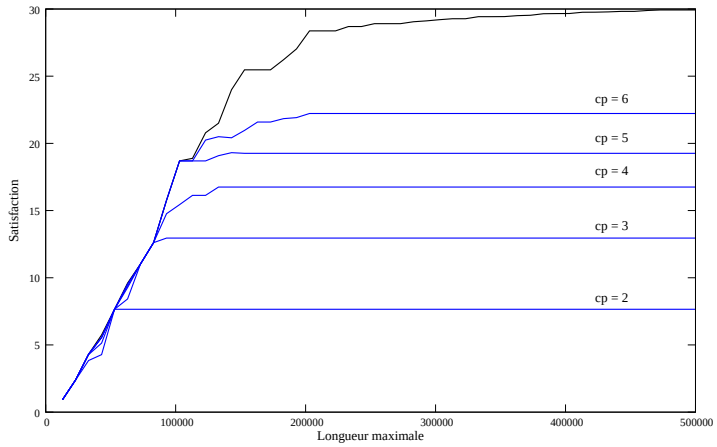
seconde avec une diversification "intelligente".

En raison du caractère aléatoire de certaines étapes de l'algorithme, chaque test a été répété 5 fois avec une valeur d'initialisation différente pour le générateur de nombres aléatoires. **Tous les résultats sont donc moyennés**, que ce soit au niveau du temps d'exécution, des différents compteurs d'itérations ou des mesures de performance (évaluation moyennée des fonctions objectif).

Note sur les graphiques

Comme au chapitre précédent, les graphiques représentent en abscisse un longueur, une borne maximale sur la taille des tournées, à comparer avec les données du graphe support ; et en ordonnée, une satisfaction dont la valeur maximale dépend du nombre de commodités envisagées. Nous avons noté *cp* le nombre de points de contrôle utilisé pour décrire la géodésique.

Instance a10e [10 sommets, peu de commodités]



La figure ci-contre (3.7) montre l'évolution de la satisfaction en fonction en fonction de la borne $p_{\max}$ sur le coût de la géodésique avec un nombre de points de contrôle *cp* donné. À des fins de comparaison, nous avons aussi représenté la solution exacte obtenue avec CPLEX.

FIG. 3.7 – Grasp simple sur a10e

On peut noter que les géodésiques se plaquent sur la courbe des solutions exactes sur un certain intervalle de p. On constate ensuite un palier : le nombre de points de contrôle de la géodésique conditionnent bien la qualité de la solution. Les résultats obtenus sont identiques aux résultats obtenus avec la méthode Tabou (voir la section 3.3.6). La différence relative de l'heuristique avec la méthode exacte est de 16% et cela malgré l'effet palier que l'on constate pour les grandes valeurs (avec 59 points d'échantillonnage). Cet écart est quasi nul sur l'intervalle où l'on ne constate pas cet effet.

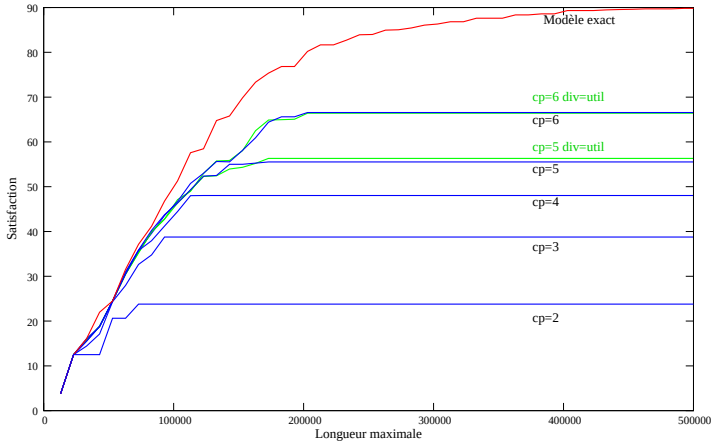
Nous comparons maintenant le GRASP avec diversification aléatoire et le GRASP avec diversification "intelligente" :

- Les tests ont été effectués sur 1770 réplications pour chaque type de diversification.
- Il n'y a pas eu de changement dans la valeur de la fonction-objectif : les deux variantes donnent les mêmes résultats.
- La méthode avec diversification "intelligente" a permis un gain sur le temps de calcul pour 65% des instances ;

- Le nombre d'itérations significatives, c'est-à-dire l'itération où la valeur optimale a été trouvée est plus faible sur 40% des instances ;

- 25 % des instances avec diversication intelligente ont un nombre global d'itérations plus faible.

Instance a10a [10 sommets, commodités point à point]



La figure ci-contre (3.7) montre l'évolution de la satisfaction en fonction en fonction de la borne $p_{\max}$ sur le coût de la géodésique. À des fins de comparaison, nous avons aussi représenté la solution exacte obtenue avec CPLEX. La figure comporte deux courbes obtenues avec diversification intelligente (repérées *div=util*).

FIG. 3.8 – Grasp simple sur a10a

Nous pouvons remarquer que les courbes obtenues par les géodésiques se "plaquent" moins bien sur la courbe exacte que pour l'instance précédente. Cela est certainement du au grand nombre de commodités : il faut plus de points de contrôle, ou alors disposer de plusieurs tournées pour améliorer l'objectif. On trouve le même type de différence relative par rapport à l'optimum (17% en moyenne et négligeable sur un sous-intervalle, $[0 : 100000]$, de $p_{\max}$).

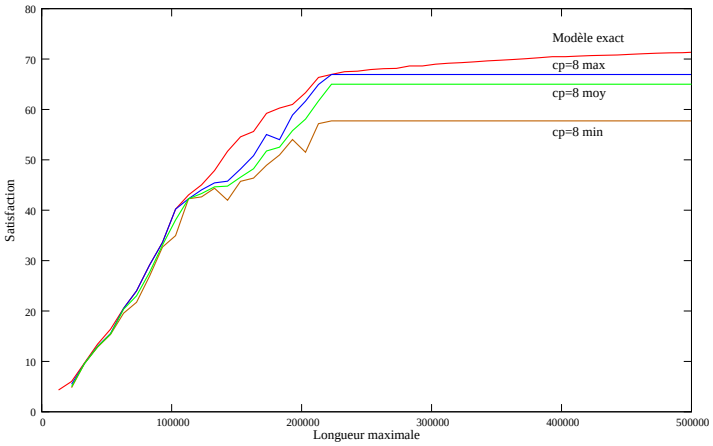
Nous comparons maintenant le GRASP avec diversification aléatoire et le GRASP avec diversification "intelligente" :

- Les tests ont été effectués sur 1770 réplications pour chaque type de diversification.

- il y a des changements sur la valeur de la fonction-objectif lors de la recherche de 5-géodésiques et de 6-géodésique (les comportements sont identiques pour des géodésiques d'ordre inférieur). Cependant, l'écart moyen est très faible (0.01%)

- la diversification intelligente permet de trouver la solution en moins d'itérations.

Instance a20e [20 sommets, peu de commodités]



La figure ci-contre (3.7) montre l'évolution de la satisfaction en fonction en fonction de la borne $p_{\max}$ sur le coût de la géodésique à 8 points de contrôle avec diversification intelligente. À des fins de comparaison, la solution exacte obtenue avec CPLEX est aussi représentée.

FIG. 3.9 – Recherche d'une 8-géodésique sur a20e

Le comportement "en escalier" de l'heuristique étant acquis, nous nous sommes limités à la recherche d'un seul type de géodésique et nous avons noté les valeurs maximales (*max*), minimales (*min*) et moyennes (*moy*) obtenues pour chaque borne.

Le fait que la courbe de la valeur maximale ne soit pas monotone croissante est un peu embêtant. Ceci peut être symptomatique d'un mauvais calibrage de la métaheuristique. Effectuer plusieurs réplications d'un même problème avec des racines différentes peut revenir en fait à accepter un plus grand nombre de diversification pour une seule réplication (si le générateur aléatoire est vraiment efficace). Cette descente signifie que le processus d'exploration a été biaisé et n'a même pas été capable de trouver une solution au moins égale.

Nous avons effectué des tests pour des géodésiques allant de 2 à 12 points de contrôle, soit environ 4200 réplications pour chaque type de diversification. Les résultats entre les deux diversifications sont très similaires : la meilleure valeur trouvée est en moyenne à 5% de l'objectif. Ce bon résultat s'explique peut-être par la disposition des commodités.

Instance a20a [20 sommets, commodités de point à point]

Nous constatons le même type de résultats que pour l'instance précédente : on reste à 15% de l'objectif. On peut noter qu'en moyenne la diversification intelligente permet d'améliorer légèrement les résultats (de l'ordre de 1%).

Remarques générales

On obtient très rapidement de "bonnes" solutions. Les recherches locales (descente) ne sont pas très longues, quelques itérations le plus souvent, et ainsi la limitation sur le nombre d'itérations pendant la recherche locale n'est pas très intéressante. L'algorithme s'arrête donc le plus souvent parce que le nombre maximal de diversifications a été atteint. Ainsi, il faut être bien attentif à la valeur de ce paramètre pour visiter suffisamment de configurations lorsque les graphes commencent à être de taille conséquente.

3.3. Métaheuristique Tabou

Le mécanisme de base de cette métaheuristique est très simple : en partant d'une solution initiale, on progresse de proche en proche en sélectionnant le meilleur voisin possible et en interdisant tout retour à une solution récemment explorée, solution que l'on qualifie alors de **tabou**. Cet algorithme n'est **pas un algorithme de descente** car la meilleure solution voisine n'est pas forcément une solution améliorante de la solution courante.

L'algorithme tabou que nous présentons est plus élaboré et se décompose en deux niveaux : la recherche "globale" pilote une recherche "locale" sous contrainte et on procède à une diversification si certains critères d'amélioration ne sont pas réunis.

3.3.1. Mémorisation d'une solution

La mise en œuvre de l'algorithme suppose que l'on soit capable de mémoriser une solution afin de ne pas la reconsidérer durant le processus de résolution. Dans notre cas, il s'agirait de stocker l'état complet d'une géodésique, c'est-à-dire de stocker la liste des points de contrôle et leurs chemins associés. Il faudrait alors disposer d'une méthode efficace de détection de l'égalité de deux géodésiques (il faut tester l'égalité de deux circuits, comme nous l'avons déjà évoqué, ce n'est pas difficile mais c'est toujours coûteux en temps de calcul).

Bien évidemment, en pratique, on ne stocke jamais complètement une solution que l'on veut bannir. On se contente d'en extraire certaines caractéristiques, que l'on nomme **caractères tabou**, qui sont à la fois faciles à identifier et porteuses d'assez d'informations pour permettre de ne pas revenir à la solution pendant un certain temps. Précisons aussi que la mémoire du processus de résolution tel que l'a conçu Glover, n'est pas infaillible : on peut même parler d'amnésie à plus ou moins long terme. Cette qualité de mémorisation est un paramètre de l'algorithme. Dans notre implémentation, le caractère tabou retenu est le fait qu'un nœud du graphe soit un point de contrôle ou pas, et ainsi nous interdisons "toute" solution qui viserait à choisir de nouveau ce sommet comme point de contrôle avant un certain nombre d'itérations. Ce caractère peut s'avérer quelquefois trop discriminatoire, c'est pourquoi on peut envisager l'utilisation du mécanisme d'**aspiration**.

3.3.2. Recherche Locale sous contrainte

On a posé une contrainte forte sur le coût de la tournée. Par choix, elle n'a pas été relâchée car le problème de la relaxation est que la projection, **le retour dans le domaine réalisable est d'autant plus dur que le problème est contraint** [GL97].

Une conséquence directe est qu'il n'est pas toujours possible d'obtenir un voisin x' lors de la recherche locale, ou alors on constate le nombre de voisins peut se réduire à une peau de chagrin (surtout lorsque cette contrainte se combine avec l'interdiction des solutions tabou). Cette situation est embarrassante dans la mesure où la métaheuristique a de bien meilleures chances de trouver une bonne solution si elle peut visiter un très grand nombre de solutions. Si on se retrouve dans ce cas de figure, on est tenté de diversifier afin de retrouver des solutions réalisables mais il ne faut pas le faire trop rapidement car la levée de caractères tabou peut permettre de découvrir de bonnes solutions. Là encore, le concept d'aspiration prend tout son sens car on peut recourir à la diversification dès lors qu'il n'y a plus de solutions réalisables améliorantes.

3.3.3. Aspiration

L'**aspiration** est **la levée du caractère tabou** d'une solution **dans certaines conditions**.

Nous avons choisi ce que Glover et Laguna appellent l'*aspiration by regional objective* [GL97, p. 53] : lors de recherche de voisinage, elle consiste à accepter une solution tabou si celle-ci améliore la solution record locale. Bien entendu, cela peut augmenter les temps de calculs car il est nécessaire d'évaluer tous les voisinages mais dans le même temps, cela réduit le nombre d'itérations pour l'obtention de la solution finale.

Notons aussi qu'il faut utiliser l'aspiration avec parcimonie dans la mesure où l'on change le comportement de l'algorithme tabou. Lorsque l'on utilise ce mécanisme, la méthode tabou se comporte localement comme un algorithme de descente, ce qui peut être parfois préjudiciable.

3.3.4. Diversification

Avec la diversification, on essaye d'explorer une nouvelle région dans l'espace des solutions. Cette étape ne fait pas partie de l'algorithme originel mais se révèle être une **extension naturelle** pour améliorer le comportement de la métaheuristique. La discussion introduite lorsque l'on a défini la diversification pour la méthode GRASP est valide et on peut se contenter d'une **version multistart** où l'algorithme de recherche tabou est relancé sans tenir compte des itérations précédentes (amnésie périodique) ; ou alors, on peut utiliser le travail déjà effectué pour la méthode GRASP et, ainsi, rendre intelligente la diversification. A ce titre, on peut alors parler d'**hybridation naturelle** entre les deux métaheuristiques GRASP et Tabou.

3.3.5. Algorithme

Paramètres :

- Q : nombre de points de contrôle de la géodésique,
- ℓ : fonction coût d'une géodésique, coût borné par $\ell_{\max}$,
- f : fonction d'évaluation d'une géodésique (par exemple, la satisfaction induite),
- $\mathcal{P}$: prédicat de test (par exemple, "inférieur ou égal") pour comparer les évaluations de deux géodésiques distinctes,
- max_g_iter : nombre maximum d'itérations globales/totales,
- max_l_iter : nombre maximum d'itérations locales,
- max_d_iter : nombre maximum de diversifications,
- max_no_nb : nombre maximum de mauvais voisinages consécutifs avant diversification,
- max_no_l_iter, max_no_g_iter : nombre maximum d'itérations (locales ou globales) consécutives sans amélioration,
- max_no_d_iter : nombre maximum de diversifications consécutives sans amélioration,
- MEM : paramètre concernant le type de mémoire (taille par exemple). Ce paramètre n'apparaît pas de manière explicite dans l'algorithme.

Lexique :

- x : solution courante,
- $\tilde{x}$: meilleure solution de la recherche locale,
- $\hat{x}$: meilleure solution de la recherche globale,
- tous les compteurs d'itérations : g_iter, l_iter, d_iter, no_nb, no_l_iter, no_g_iter, no_d_iter dont on connaît les valeurs maximales.

La liste tabou T est mise à jour à chaque itération locale. Si la mémoire est limitée, cela permet de rendre certains mouvements à nouveau licites (les mouvements les plus anciens sont retirés de la liste). Lors de la phase de recherche locale, on génère le voisinage de la solution courante : toutes les solutions de ce voisinage sont réalisables mais ne sont pas forcément améliorantes par rapport à la solution courante (c'est la grande différence avec un mécanisme de descente classique).

Algorithme 6 : Recherche Tabou avec diversification et contrainte forte sur la recherche locale

```
⇝ initialisations
générer aléatoirement une solution initiale réalisable x_0 (une Q-géodésique)
initialiser la liste tabou : T ← ∅
x̂ ← x̃ ← x ← x_0, g_iter ← d_iter ← no_g_iter ← no_d_iter ← 0

⇝ itérations au niveau global
tant que critère_1 non satisfait faire
    ⇝ itérations au niveau local
    l_iter ← no_l_iter ← no_nb ← 0
    tant que critère_2 non satisfait faire
        incrémenter les compteurs : g_iter, l_iter, no_g_iter et no_l_iter
        sélectionner le meilleur voisin x' ∈ V(x), tel que { x' ∉ T ; ℓ(x') ≤ ℓ_max
        si x' existe alors
            si P(f(x'), f(x)) alors
                x̃ ← x'
                no_l_iter ← no_g_iter ← 0
            fin si
            insérer x dans T, no_nb ← 0
            x ← x'
        sinon incrémenter no_nb
        mettre à jour T
    fin tant que
    si P(f(x̂), f(x̃)) alors
        x̂ ← x̃
        no_g_iter ← no_d_iter ← 0
    sinon incrémenter no_g_iter et no_d_iter

    ⇝ diversification
    incrémenter g_iter et d_iter
    générer une nouvelle solution réalisable x
    T ← ∅
fin tant que
```

Dans l'algorithme, nous n'avons pas explicité les critères d'arrêt des boucles, repérés $critère_1$ et $critère_2$. Nous le faisons donc maintenant.

Critère 1 : C'est le critère d'arrêt de la recherche globale. Cette recherche se termine si l'un des cas suivants se produit :

- un certain nombre d'itérations globales a été dépassé [max_g_iter],
- un certain nombre de diversifications a été atteint [max_d_iter],
- la valeur record global n'a pas été améliorée pendant un certain nombre d'itérations [max_no_g_iter],

- la valeur record global n'a pas été améliorée pendant un certain nombre de diversifications [max_no_d_iter].

Critère 2 : C'est le critère d'arrêt de la recherche locale. On sort de cette recherche si l'un des cas suivants se produit :

- on a déjà effectué un certain nombre d'itérations locales (et ce, même s'il est toujours possible d'améliorer la solution courante) [max_l_iter],
- la solution courante n'a pas été améliorée pendant un certain nombre d'itérations [max_no_l_iter],
- un certain nombre de voisinages non exploitables a été généré [max_no_nb],
- le nombre d'itérations globales a été dépassé [max_g_iter].

Si l'on veut prendre en compte la notion d'**aspiration**, il est nécessaire d'adapter l'algorithme précédent au niveau de la recherche locale. On lève le caractère tabou d'une solution réalisable x' si celle-ci est meilleure qu'une solution record $\tilde{x}$, la meilleure solution de la recherche locale.

Algorithme 7 : Modification de la phase de recherche locale pour prendre en compte la notion d'aspiration

```
x' ← ∅ // on cherche x' meilleur voisin de x donné
pour tout v ∈ V(x) faire
    si v ∈ T alors
        si P(f(v), f(x̃)) alors
            x' ← v // aspiration
        fin si
    sinon
        si P(f(v), f(x')) alors  x' ← v
    fin si
fait
```

3.3.6. Expérimentations

Présentation des tests

Nous reprenons les instances de test décrites an annexe page 155 que nous avons déjà utilisées dans la mise en œuvre des modèles au chapitre 2 (section 2.5 page 55).

Au travers de ces différentes instances, nous devons tester et évaluer l'influence de différents paramètres :

- la résolution Tabou "simple",
- l'influence du mode de déplacement : PCC ou réaliste,
- l'influence du type de diversification : la diversification aléatoire multistart ou bien la diversification "intelligente" (hybridation avec le GRASP),
- l'influence de la notion d'aspiration.

Instance a10e [10 sommets, peu de commodités]

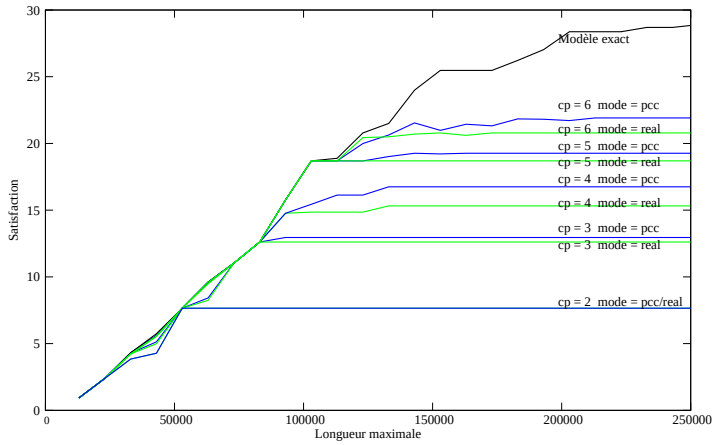


FIG. 3.10 – Tabou simple sur a10e

La figure ci-contre (3.10) montre l'évolution de la satisfaction en fonction en fonction de la borne $p_{\max}$ sur le coût de la géodésique. Nous disposons de données sur l'intervalle [0 : 600000] mais nous nous sommes limités à l'intervalle [0 : 250000]. Cela suffit pour illustrer le comportement de la métaheuristique. Cette figure regroupe plusieurs tests. De haut en bas, on trouve la solution optimale donnée par le modèle exact, puis la solution pour le mode PCC avec 6 points de contrôle, la solution pour le mode réaliste (paresseux) avec six points de contrôle et ainsi de suite...

On remarque que le mode PCC donne, en général, une meilleure évaluation que le mode réaliste (les évaluations sont égales pour les géodésiques à 2 points de contrôle).
À un nombre de points de contrôle fixé, il existe un pallier en $p$max pour lequel, on ne trouve plus de meilleure solution. Notons que ces courbes ne sont pas monotones car nous avons pris une moyenne des résultats. La courbe est croissante si l'on prend les meilleures satisfactions et non plus les moyennes.
Plus le nombre de points de contrôle est élevé, plus on arrive à approcher la solution optimale, donc on est d'ailleurs très proche pour des valeurs de $p$max relativement petites. Pour améliorer le comportement de l'heuristique sur des bornes de coût plus importantes, il faut augmenter le nombre de points de contrôle ou bien utiliser plusieurs tournées. **Le nombre de points de contrôle conditionne bien la qualité de la solution.**

Sur un réseau aussi petit, la solution optimale est trouvée très rapidement. La métaheuristique effectue un très grand nombre d'itérations pendant lesquelles elle cherche à encore améliorer la solution courante.

On constate un nombre important d'échecs lors de la recherche locale sous contrainte, c'est-à-dire que l'on ne dispose pas d'un solution voisine réalisable pour continuer le processus d'exploration. Comme le graphe est très petit, on retombe en fait sur des configurations tabous. Plusieurs solutions sont alors envisageables : lever le caractère tabou grâce au mécanisme d'aspiration, enlever les solutions les plus anciennes de la mémoire de la métaheuristique pour essayer de trouver des solutions réalisables (cette opération peut être effectuée un certain nombre de fois) et en dernier, diversifier. Bien entendu, ces mécanismes sont en compétition : une diversification trop rapide, sans attendre la levée de certains caractères tabous, peut être une maladresse. L'utilisation intensive de l'aspiration force la méthode tabou à adopter le comportement d'un algorithme de descente.

Instance a10a [10 sommets, commodités point à point]

Nous avons testé ce réseau dans les mêmes conditions que le réseau précédent, c'est-à-dire avec 1770 réplications pour chaque famille de tests.

Si l'on s'intéresse aux modes de déplacement des produits, le mode réaliste apporte une légère dégradation de la valeur obtenue avec le mode de déplacement par plus courts chemins.

La diversification intelligente (issue de la méthode GRASP) permet d'obtenir de meilleurs résultats plus rapidement (aussi bien en temps de calcul que du nombre d'itérations).

L'utilisation de la notion d'aspiration n'a pas eu d'effet sur cette instance, nous avons trouvé les mêmes résultats en moyenne.

Si l'on compare nos implémentations de la méthode GRASP et de la méthode Tabou, nous constatons que la méthode GRASP est toujours plus rapide mais que la méthode Tabou donne en moyenne de meilleurs résultats.

Instance a20a [20 sommets, commodités point à point]

Nous avons réalisé des tests sur l'instance a20a avec des géodésiques allant de 2 à 12 points de contrôle, cela nous a donné environ 800 points d'échantillonnage et plus de 3300 réplications au total pour chaque type de test. Nous nous sommes contenté du mode réaliste. Le meilleur objectif obtenu est en moyenne à 18% du résultat optimal obtenu en mode PCC.

La diversification intelligente permet d'obtenir de meilleurs résultats sur 50% des réplications et plus rapidement (90% des réplications).

L'utilisation de la notion d'aspiration n'a pas apporté de gain probant.

Instance cf992 [92 sommets, 992 commodités]

La figure (3.11) montre l'évolution des différents solutions records mémorisées en fonction du nombre d'itérations effectuées par la méthode tabou. On recherche une 4-géodésique avec une contrainte de taille sur le réseau cf992. On peut noter différents paliers : le premier, le plus visible est la convergence du record global vers son optimum. On peut ensuite remarquer plein de petits paliers généralement suivis par des traits verticaux : il s'agit de l'évolution de la solution record locale. Chaque palier représente un optimum local donc l'algorithme tabou essaie de sortir en provoquant une diversification (un trait vertical). Le nuage de points représente simplement les différents valeurs des solutions à un coût donné.

La solution renvoyée finalement par l'algorithme est trouvée rapidement mais l'algorithme effectue un grand nombre d'itérations et de diversifications afin de s'assurer qu'il ne trouve pas de meilleure solution.

3.4. Accélérateurs de calcul

Un paramètre non explicité dans les algorithmes de principe des deux métaheuristiques que l'on vient de donner est le prédicat $\mathcal{P}$ permettant de comparer à tout moment deux solutions réalisables x et x' afin de savoir laquelle des deux est la meilleure.

L'évaluation de la qualité d'une solution x peut être très coûteuse en temps machine. Prenons, par exemple, le cas de notre problème de transport sur les réseaux de mobilité avec demande élastique, la qualité d'une solution se mesure grâce à sa qualité de service. Le problème consiste à maximiser une certaine fonction Z où $Z = QoS(x)$ est la qualité de

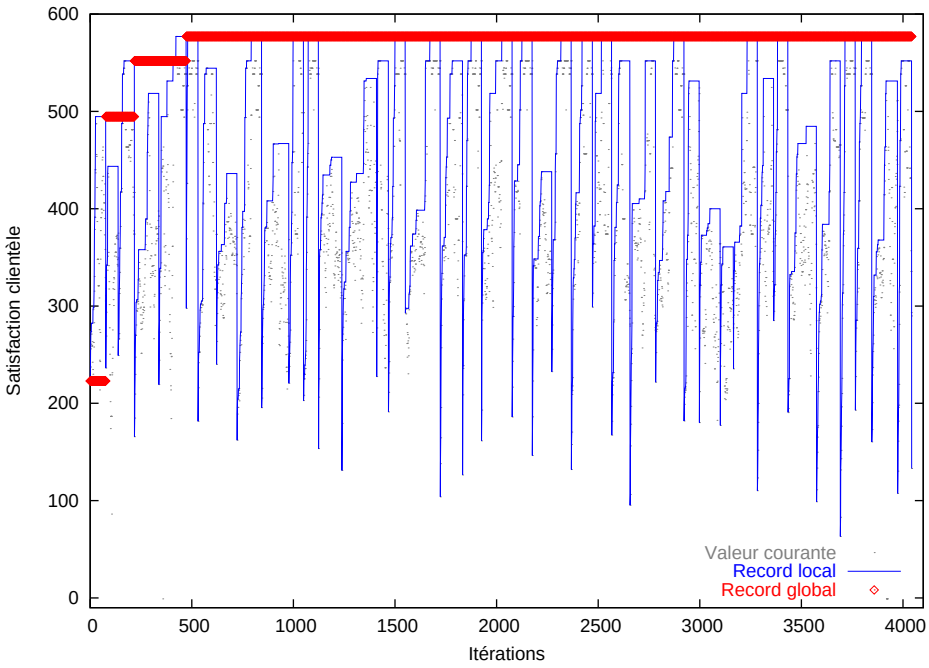


FIG. 3.11 – Résolution d'une instance de cf992

service d'une solution réalisable x. Nous avons envisagé différentes fonctions de qualité de service, par exemple, $Qos(x) = \sum_{k \in K} D^k \Phi^k(t^k)$ ou $QoS(s) = \min_{k \in K} D^k \Phi^k(t^k)$ avec des fonctions de satisfaction Φ^k pour chaque commodité k et des temps de parcours induits t^k (voir les sections 2.4.1 et 2.4.3).

Outre le temps de calcul des différentes fonctions Φ^k, ce qui peut inclure le calcul de certains paramètres implicites mais nécessaires (dans notre cas, expression du routage et évaluation des temps de parcours induits), la fonction objectif dépend du nombre de commodités du système. Ainsi, plus ce nombre est élevé, plus le temps de calcul de la fonction objectif est long. Or, cette fonction est utilisée très souvent dans le processus de résolution. En fait, dès qu'une nouvelle solution est découverte et doit être évaluée (tout voisin réalisable du GRASP et tout voisin non interdit de la méthode tabou). Ainsi, même sur des graphes de taille raisonnable (un graphe avec une centaine de nœuds par exemple), on atteint déjà plusieurs centaine de milliers d'évaluations.

Il est donc intéressant d'essayer d'**accélérer certaines parties de l'algorithme de résolution, notamment la fonction objectif dont l'utilisation est intensive**.

Bien entendu, s'il était vraiment possible d'accélérer la fonction objectif par des astuces d'implémentation, cela ne réglerait pas le problème du nombre potentiellement grand d'appels à cette fonction. Le problème est donc bien plus profond. On peut donc se poser la question, légitime, de savoir s'il serait possible, dans certaines situations, de substituer la fonction objectif originale par une autre fonction, que l'on appellerait fonction auxiliaire, qui serait plus rapide à calculer. Il faut cependant être vigilant dans cette substitution dans la mesure où l'**on ne doit pas changer de problème**.

De même, si l'on accepte un tel changement de fonction objectif pour une ou plusieurs étapes du processus de résolution, il faut accepter que l'**accélération potentielle se fasse**

au détriment de la qualité de la solution trouvée sans accélération.

3.4.1. Fonction-objectif auxiliaire

Pour déterminer cette fonction auxiliaire, nous nous sommes attachés à favoriser des propriétés que l'on retrouve "souvent" dans les solutions optimales comme des critères de forme ("convexité de la solution") ou bien comme des mesures d'intérêt des arcs.

Prenons un exemple concret. Si la qualité de service est évaluée suivant un critère temps, l'acheminement d'un produit d'une origine à une destination donnée sera optimal s'il est réalisé suivant un plus court chemin dans le réseau. Si l'on compte le nombre de fois où un arc est présent dans un plus court chemin et que l'on fait cela pour toutes les commodités du réseau, on peut se dire, qu'intuitivement, une solution pour la tournée devrait comporter un certain nombre de ces arcs à fort coefficient.

D'un autre côté, lorsque l'on connaît la tournée effectuée par les conteneurs, on peut se dire que le trajet des produits sera d'autant plus rapide que le parcours dans la tournée sera efficace. Par exemple, si l'on effectue le trajet sur la tournée d'un point A à un point B, ce trajet doit être le plus proche possible du plus court chemin entre ces deux points (finalement, le trajet sur la tournée entre A et B et la corde entre ces deux points doivent être égaux).

Essayons maintenant de formaliser ces intuitions. Supposons que l'on dispose d'une famille de poids $(w_e)_{e\in A}$ positifs, représentative de l'intérêt des arcs par exemple. Alors, on définit la fonction Z' suivante pour une Q-géodésique notée Γ :

$$Z' = \underbrace{\sum_{e\in\Gamma} w_e}_{(1)} + \underbrace{\sum_{q=1}^{Q} \left[\Phi(t_q, t_q^*)(\sum_{e\in\Gamma_q} w_e) \right]}_{(2)}$$

où

- les Γ_q sont les chemins parcourus sur la géodésique du milieu du q$^{\text{ème}}$ PCC au milieu du PCC suivant (les carrés à la figure (3.12)). Attention, il ne s'agit pas des chemins géodésiques, car par définition, $t_q = t_q^*$.
- t_q est le temps de parcours d'un tel chemin en vitesse rapide,
- t_q^* est le temps optimal (de marche à pied) pour se rendre entre les deux extrémités de ce chemin.

Le premier terme (1) vise à choisir des **arcs intéressants**, relativement à la famille $(w_e)_{e\in E}$ de poids sur les arcs : on est donc amené à choisir des arcs dont le poids est important ou alors à choisir un très grand nombre d'arcs pour que la somme soit maximale.

Le deuxième terme (2) cherche à **accélérer le trajet sur la géodésique**, en vérifiant que la corde entre deux milieux de PCC consécutifs n'est pas trop performante par rapport au chemin sur la tournée. Ce critère permet plutôt de diminuer la taille (et la forme) de la tournée. Nous avons introduits les milieux des chemins de PCC car cela n'aurait pas eu de sens d'utiliser les points de contrôle (le chemin entre deux points de contrôle, un PCC, est déjà une corde).

Sur la figure (3.12), les cercles représentent les points de contrôle, les carrés les milieux des chemins géodésiques. On a représenté en pointillé la corde associée à Γ_1 entre le milieu du PCC associé au premier point de contrôle et le milieu du PCC suivant.

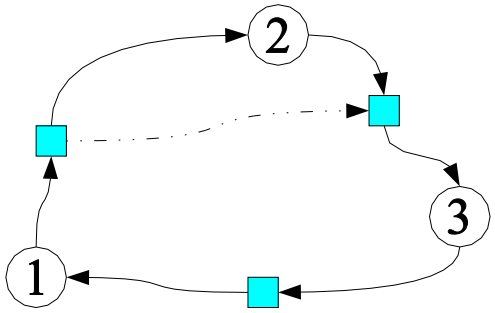


FIG. 3.12 – Points particuliers de la géodésique

Nous avons déterminé le milieu d'un chemin géodésique, non pas en terme de distance, mais en terme de nombre de sommets entre l'origine et la destination de ce chemin géodésique. On appelle M_i le milieu du i$^{\text{ème}}$ chemin géodésique :

$$M_i = \begin{cases} PC(i) \text{ si } PC(i) = PC(i+1) \\ origine\left(Arcs_{PCC_i}[\frac{nb\_arcs PCC_i}{2}]\right) \text{ sinon} \end{cases}$$

où $PC(i)$ est le i$^{\text{ème}}$ point de contrôle, PCC_i le i$^{\text{ème}}$ PCC.

On peut encore se poser la question de savoir si l'on peut déterminer une bonne famille d'indicateurs, ou poids, sur les arcs pour être sûr que cette famille d'arcs soit intéressante sans la mettre à jour trop souvent.

On propose, par exemple, de mesurer l'utilisation idéale des arcs, c'est-à-dire compter le nombre de fois où un arc se trouve sur un PCC entre une origine et une destination d'une commodité. Une telle famille de poids est statique. Si l'on cherche la meilleure tournée qui passe par ces arcs, on se ramène à un problème de type postier rural, qui est aussi un problème difficile.

Si la famille de poids est dynamique (si, par exemple, elle est mise à jour à chaque amélioration de la valeur record globale), il faut faire attention aux pertes de temps induites par le calcul des poids.

On peut aussi essayer de mesurer l'utilité des arcs en employant un algorithme de k plus courts chemins. Pour une commodité donnée, on associe aux arcs du plus court chemin une certaine utilité, on associe ensuite une utilité inférieure au chemin qui est juste un peu moins bon que le plus court chemin et ainsi de suite.

3.4.2. Utilisation des accélérateurs dans les métaheuristiques

Nous avons décrit précédemment le mécanisme des algorithmes GRASP et Tabou. Peut-on introduire facilement le concept de fonction auxiliaire dans ces deux méthodes sans changer de problème ?

On note dans ce paragraphe Z la fonction objectif du problème d'origine (par exemple $Z = QoS(x)$) à maximiser. On note Z' la fonction auxiliaire que l'on veut aussi maximiser.

Méthode GRASP

La méthode GRASP est une méthode de descente avec relance "intelligente" périodique. Pour pouvoir substituer la fonction objectif initiale par une fonction auxiliaire, il est nécessaire

que les comportements soient très proches car le voisin d'une solution donnée est une autre solution réalisable **améliorante**. Ce n'est malheureusement pas forcément le cas : pour que les différents objectifs coïncident un tant soit peu et avoir localement $Z \approx Z'$, il faut passer beaucoup de temps sur la mise à jour des différents paramètres (w_e) de la fonction auxiliaire. Dans ce cas-là, il vaut mieux utiliser la fonction originale, ce sera long, sûrement, mais on aura optimiser le bon objectif.

Algorithme 8 : GRASP classique sans fonction auxiliaire

tant que non arrêt **faire**
  diversifier
  descente ($\max Z$)
fin tant que

Algorithme 9 : GRASP avec fonction auxiliaire

tant que non arrêt **faire**
  diversifier
  mettre à jour (w_e)
  descente ($\max Z'$)
fin tant que

Si les (w_e) sont des constantes topologiques, elles ne seront calculées qu'une seule et unique fois, au début de l'algorithme.

Méthode Tabou

Le processus de choix d'une nouvelle solution de la méthode Tabou est plus souple que celui de la méthode GRASP dans la mesure où la méthode Tabou n'est pas un algorithme de descente : le voisin retenu n'est pas forcément un voisin améliorant, il s'agit juste du meilleur voisin au sens de la fonction d'évaluation retenue.

Supposons que l'on substitue la fonction originale Z à la fonction auxiliaire Z' lors de l'exploration du voisinage et uniquement lors de cette étape. Le voisin retenu peut alors être évalué grâce à la fonction Z et le reste de l'algorithme est inchangé. On n'a donc pas changé l'objectif global du processus de résolution et on résout donc bien le même problème.

L'exploration du voisinage d'une solution donnée, anciennement très consommateur de ressources de calcul, est maintenant bien plus rapide. Il faut encore vérifier expérimentalement que les familles de poids (w_e) sont bien choisies et que le processus global est efficace.

Algorithme 10 : Adaptation de l'algorithme Tabou pour prendre en compte la notion de fonction objectif auxiliaire

Initialiser solution x et (w_e)
tant que non(critère_fin) **faire**
  déterminer $x' = \max_{y \in V(x)} Z'(y)$
  si $(x \neq \emptyset)$ **ET** non(critère_fin) **ET** non (critère_diversification) **alors**
    $x \leftarrow x'$
    calcul de $Z(x')$, mise à jour du record
  fin si
  sinon si non (critère_fin) **alors** diversification, mise à jour des (w_e)
fin tant que

3.4.3. Expérimentations

Nous ne donnerons pas dans cette section de résultats concernant la méthode GRASP. En effet, comme il s'agit d'une méthode de descente, nous n'avons pas été capables de trouver une bonne famille de poids topologiques qui satisfasse aussi la fonction objectif initiale.

Nous allons mesurer l'impact du changement de la fonction d'évaluation dans le processus de recherche de voisinage de la méthode Tabou. Nous allons comparer les qualités des solutions, le nombre d'itérations et le gain ou la perte de temps entre les deux approches.

Sur l'**instance a10e**, les résultats obtenus au niveau de l'objectif final sont identiques à ceux obtenus avec la méthode tabou simple. Le gain moyen en vitesse est énorme, de l'ordre de 70%.

Sur l'**instance a20a**, les résultats obtenus au niveau de l'objectif final sont identiques à la méthode tabou simple. 75% des instances sont obtenues plus rapidement.

Sur les petites instances, nous obtenons les mêmes résultats en moyenne que ceux obtenus avec une résolution par l'algorithme simple. La qualité de la solution finale est dégradée avec des réseaux de plus grande taille.

3.5. Extension de voisinage

3.5.1. Présentation

Nous avons défini la géodésique "de base" comme un ensemble de sommets ordonnés reliés par des plus courts chemins. Dans certains cas, nous n'avons pas le choix : il n'existe qu'un seul plus court chemin entre deux points de contrôle, on peut donc dire que le voisinage défini par cet opérateur de changement de chemin est de taille vraiment restreinte. Cela peut s'avérer gênant dans la mesure où les méthodes de résolution fondées sur la recherche locale ont **besoin de disposer de voisinages importants et variés** pour avoir plus de chances de trouver une bonne solution.

A la section (3.1.5) de ce chapitre, nous avons énuméré plusieurs manières d'étendre la définition de la géodésique : la notion de chemins entre deux points de contrôle est élargie pour englober des chemins autres que les PCC. L'idée sous-jacente est d'augmenter naturellement la taille des voisinages disponibles.

Nous proposons d'utiliser une liste de chemins entre deux points de contrôle tirée de la résolution de problèmes de K plus courts chemins. Nous présentons un algorithme de K-PCC en annexe, au paragraphe (A.7).

3.5.2. Expérimentations

Qu'apporte donc cette extension par rapport aux algorithmes originaux ?

Au niveau de la fonction objectif tout d'abord, si l'on se place dans les mêmes conditions, c'est-à-dire avec la même borne sur le coût $p_{\max}$ et le même nombre de points de contrôle, on constate une amélioration de la fonction-objectif. En fait, on trouve la même solution que pour la méthode classique mais avec quelques points de contrôle supplémentaires. L'extension de voisinage permet d'obtenir donc des **solutions bien meilleures avec un nombre de points de contrôle plus faible**.

Cependant, cette amélioration de la fonction objectif a un coût : le temps d'exploration des différents voisinages est bien plus important et au final, la résolution est plus longue.

Nous retrouvons ces constatations pour l'**instance a10a** où tant au niveau des temps de calcul que du nombre d'itérations, cette extension est plus lente mais les objectifs finaux sont meilleurs, évidemment quand on n'a pas pu atteindre l'ojectif optimale avec la méthode standard. Ce résultat vient du fait que l'on limite le nombre de points de contrôle et la relation entre ces points de contrôle est plus lâche dans l'extension, on trouve des chemins que l'on n'aurait pas pu retrouver autrement.

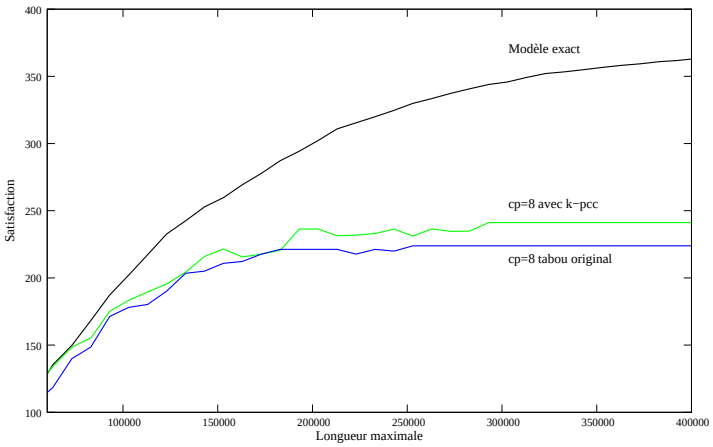


FIG. 3.13 – Méthode des K-PCC sur a20a

Prenons l'exemple ci-contre de la recherche d'une 8-géodésique sur l'instance a20a avec l'algorithme tabou en mode réaliste. On a représenté la solution optimale en mode PCC obtenue par CPLEX, la solution obtenue avec l'algorithme Tabou de base et enfin, la solution obtenue avec le Tabou et l'extension de la définition de la géodésique. On voit clairement le gain obtenu en terme de satisfaction avec l'extension par rapport à l'algorithme d'origine. Seul bémol, ce gain de performance se fait au détriment du temps de calcul, le temps passé à explorer un plus grand voisinage.

3.6. Résolution du problème multitournée

Tous les concepts que nous avons introduits jusqu'à maintenant dans ce chapitre, ont été illustrés en résolvant un problème monotournée. Nous nous intéressons maintenant au problème avec plusieurs tournées.

Nous allons décrire les modifications à effectuer pour que le travail sur une tournée puisse servir pour un nombre de tournées fixé. En particulier, nous allons présenter comment adapter les algorithmes et comment poser la problématique sous-jacente qui est l'évaluation de la satisfaction liée au routage proposé pour les porduits.

3.6.1. Routage des produits et évaluation de la satisfaction induite

Dans le cas multitournée, proposer un routage des produits, c'est-à-dire déterminer les chemins que ceux-ci doivent emprunter est une véritable problématique et pourtant il est au cœur du mécanisme d'évaluation des commodités et donc du calcul de la satisfaction induite par le routage telle que nous l'avons définie. Pour mémoire, on évalue les temps de parcours et l'on compare ces temps par rapport à des temps de référence.

Là encore, nous allons distinguer deux cas : l'évaluation par plus court chemin et l'évaluation "réaliste". Nous allons étudier le problème des connexions suivant les deux hypothèses de déplacement.

Évaluation par plus court chemin

Comme son nom l'indique, nous allons évaluer le déplacement des roduits en appliquant des algorithmes de plus courts chemins.

Si le nombre de connexions (changements de ligne) n'est **pas borné**, on se retrouve exactement avec les mêmes hypothèses que pour les modèles linéaires donnés au chapitre 2. Ces modèles bien qu'extrêmement simplistes donnent une borne inférieure sur le temps de trajet des différentes commodités. Les satisfactions sont alors idéales et ne peuvent pas refléter un comportement réel. Il suffit d'évaluer directement les temps des commodités en appliquant un algorithme de PCC tout simple (il n'est pas nécessaire de calculer explicitement les chemins).

En revanche, si on veut **limiter le nombre de connexions**, l'évaluation est bien plus difficile. Dans le cas le plus simple, on considère des temps d'attente connus aux nœuds du graphe. On peut alors espérer que ceux-ci sont dissuasifs et ainsi que le nombre de changements sera naturellement limité. Dans le cas contraire, il faut expliciter les chemins des produits et identifier les connexions pour pouvoir contraindre leur nombre. Cette analyse engendre un surcoût et l'évaluation qui est une opération cruciale des métaheuristiques va faire exploser le temps d'exécution des algorithmes. On peut envisager différentes possibilités :

- appliquer un algorithme de PCC sous contrainte ou bi-objectif (en comptant le nombre d'arcs de montée dans un conteneur dans un graphe où la couche lente et les couches rapides sont dissociées) ;
- un algorithme de k-PCC incrémental avec analyse des chemins.

Évaluation "réaliste"

Proposer un routage "réaliste" des produits est là aussi très difficile. Nous avons fondé cette notion sur le fait que l'opération de descente d'un conteneur pour y remonter n'est pas toujours souhaitable (un comportement paresseux pour un usager ou des contraintes de manutention pour le chargement/déchargement d'une marchandise). Alors que peut-on appeler réellement un déplacement réaliste ?

- un déplacement avec un **nombre minimal** de changements de ligne, ou alors un nombre borné ? En général, effectuer une ou deux correspondances est une bonne moyenne mais le nombre maximal de changements peut aussi dépendre de la longueur du trajet.
- un déplacement où le temps de déplacement en vitesse lente est minimal ?

Bien entendu, le comportement réel de déplacement intègre de manière plus ou moins inconsciente ces deux paramètres et bien d'autres encore. Toute connexion ou changement de ligne doit apporter un gain non négligeable pour être réalisé.

Nous allons nous contenter de respecter le principe qui veut qu'une fois que le produit se trouve dans un conteneur, celui-ci n'en redescendra pas s'il doit le reprendre ultérieurement. De ce point de vue, nous allons "lisser" le trajet de l'usager.

Nous avons utilisé un formalisme issu des expressions régulières [FRI03], notion développée pour les théories des automates et des langages formels, pour découper le routage d'un produit en tronçons réalisés en vitesse lente p et en tronçons dans un conteneur sur une ligne donnée. On note le tronçon b_i pour la ligne i.

On peut dire alors que le trajet d'un produit sera de la forme :

$$(p \mid b_i)^*$$

x^+	x est utilisé au moins une fois
x^*	x est utilisé au moins une fois ou pas du tout
$x?$	x est utilisé zéro ou une fois
$a \mid b$	a OU b
$a \ . \ b$	a suivi de b

TAB. 3.1 – Lexique pour les expressions régulières

On peut se poser la question pour la méthode à suivre lorsque le nombre maximal de connexions est limité. Prenons l'exemple de n tournées avec un seul changement autorisé, on peut essayer d'évaluer toutes les possibilités (au nombre de $C_n^2 + n + 1$) qui sont de la forme : $\{p\}, \{p?\ b_i\ p?\}, \{p?\ b_i\ p?\ b_j\ p?, i \neq j\}$

De plus, en pratique, il est possible d'éliminer les tournées qui se trouvent trop loin des points de départ des trajets (n'oublions pas que le trajet final ne peut être plus long que le trajet uniquement piéton).

Peut-on trouver une méthode générale de détermination d'un bon trajet ?

On propose d'analyser un chemin donné en appliquant un algorithme de PCC et de le modifier pour le rendre conforme à notre vision d'un chemin réel.
Prenons un premier exemple, figure (3.14) :

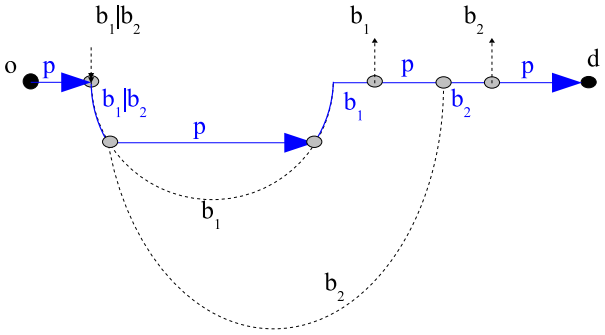


FIG. 3.14 – Premier exemple de multitournée

L'algorithme nous donne les arcs du chemin de PCC et certains arcs sont supports de plusieurs tournées. Le PCC est le suivant : $p\ (b_1 \mid b_2)\ p\ b_1\ p\ b_2\ p$.

On a alors deux possibilités. On peut se contenter de gommer la sortie de b_1, ce qui donne le trajet suivant : $p\ b_1\ p\ b_2\ p$, mais on peut aussi considérer l'autre alternative, c'est-à-dire $p\ b_2\ p$. Il suffit de prendre la meilleure évaluation (pour cela il faut que la connexion soit acceptée).

Considérons un deuxième exemple, donné à la figure (3.15). Le routage se fait suivant le plus chemin comme suit : $p\ (b_1|b_2)\ p\ b_1\ p$.

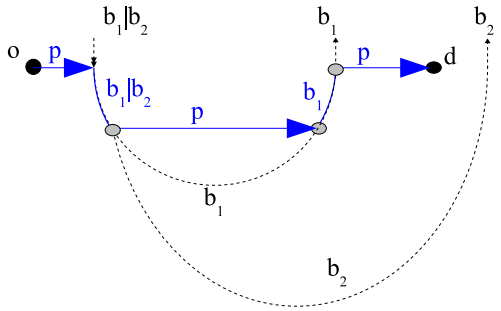


FIG. 3.15 – Multitournée 2a

Si l'on gomme la première sortie de conteneur, il faut choisir entre deux alternatives : le trajet uniquement sur b_1, comme le montre la figure (3.16), ou alors, au contraire, uniquement sur b_2, figure (3.17).

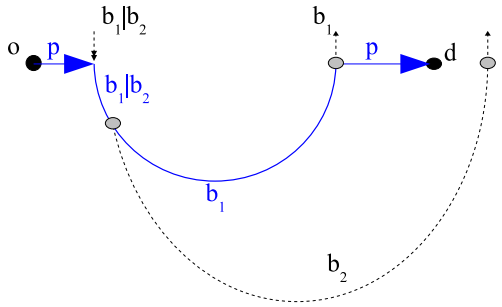


FIG. 3.16 – Multitournée 2b

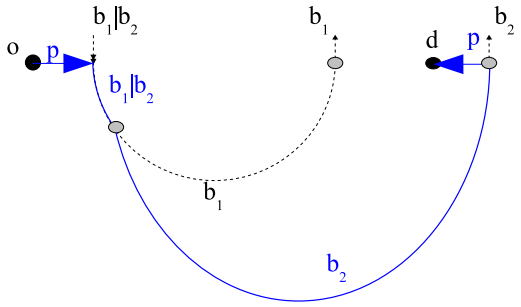


FIG. 3.17 – Multitournée 2c

Prenons le dernier exemple ci-contre, figure (3.18). Le trajet du produit considéré entre o et d est : $p\ b_1\ p\ b_1\ p$. Si on enlève la section lente entre les deux sections rapides b_1, le trajet devient alors $p\ b_1\ p$.

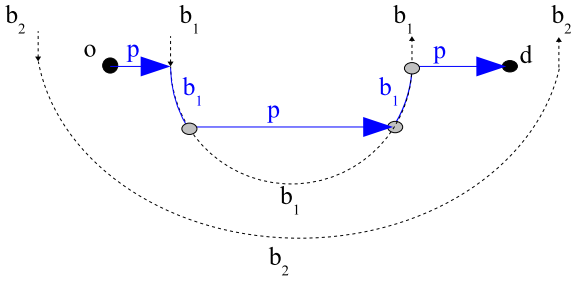


FIG. 3.18 – Multitournée 3a

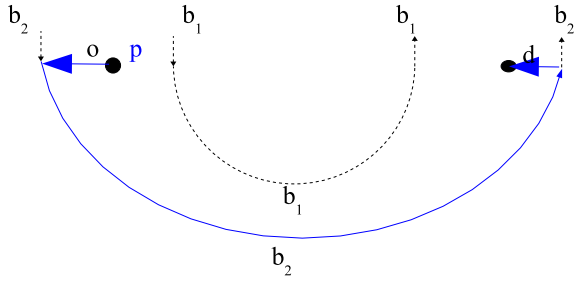


FIG. 3.19 – Multitournée 3b

Il est alors possible que le trajet empruntant la ligne b_2, $p\ b_2\ p$ devienne plus intéressant que le trajet qui emprunte uniquement la ligne b_1. L'algorithme de lissage du plus court chemin doit être capable de retrouver une telle configuration.

Le nombre de configurations à traiter peut donc être très important, nous allons nous limiter à quelques cas. Nous proposons un algorithme itératif pour chaque commodité sur un réseau multicouche. On propose de calculer le plus court chemin et de lisser progressivement celui-ci. On va identifier une configuration que l'on veut éviter absolument (une par exemple qui permet la descente puis la montée dans la même ligne). On enlève du graphe les arcs qui permettent la connexion identifiée et on relance le processus.

On va donc chercher à éliminer les configurations suivantes :

$$b_i\ (\ p \mid b_j\ )^*\ b_i, \quad \forall j \neq i$$

On enlèvera la configuration la plus intéressante à priori, c'est-à-dire celle qui dégrade le moins la valeur du plus court chemin courante.

La complexité du processus de lissage de chemin est donc en $O(m)$ de celle d'un plus court chemin (En effet, dans le pire des cas, on n'enlève qu'un seul arc à la fois et cette opération ne peut être répétée plus du nombre d'arcs dans le graphe, m, fois.

Algorithme 11 : Lissage d'un plus court chemin

Entrée : une commodité (o, d)
Retourne : un trajet de o à d "réaliste"
Calculer le PCC de o à d
tant que il existe une configuration interdite **faire**
  enlever les arcs de la configuration exhibée la plus intéressante
  mettre à jour le PCC
fin tant que

Nous nous limitons à cette **version gloutonne**. On pourrait tout à fait envisager une version qui explore toutes les combinaisons possibles et ne garder que les meilleures. Cependant, le temps de calcul nécessaire pour la détermination d'un seul trajet, et donc d'un très grand nombre de commodités, peut dissuader d'utiliser une telle méthode dans la mesure où cet algorithme sera appelé très souvent dans les algorithmes de résolution.

3.6.2. Adaptation des algorithmes de résolution

Si on prend le point de vue de la compagnie qui gère le transport, il est clair que le nombre de tournées doit être **minimal**. En effet, l'installation d'une nouvelle ligne génère un coût fixe qu'il faut être sûr de pouvoir amortir. Cette réalité est d'ailleurs souvent reprise dans les problèmes classiques de tournées de véhicules.

De la même façon, la longueur des tournées doit être impérativement **bornée** que ce soit pour des raisons budgétaires, humaines, sociales ou techniques.

On veut que les algorithmes que l'on a employés soient capables de gérer plusieurs tournées. Nous devons d'abord choisir si le nombre de tournées est déterminé à l'avance ou si ce nombre peut évoluer en cours d'exécution (avec, éventuellement, avec une borne sur ce nombre). Nous nous plaçons dans le cas où nous laissons choisir à l'organe décisionnel le nombre de tournées.

Nous allons donc gérer, non plus une seule tournée dont la taille est éventuellement bornée, mais plutôt un nombre **fixe** de tournées de taille bornée.

Comment gérer la diversification ?

La diversification a pour but de générer de nouvelles tournées dans l'optique d'explorer une zone nouvelle de l'espace des solutions.

Si la diversification est aléatoire, cela ne pose pas de problème, il suffit de lancer plusieurs fois le processus de génération d'une tournée, en vérifiant toutefois que les tournées sont bien distinctes. Nous nous contenterons de cette approche.

Si l'on veut utiliser à bon escient les informations glanées lors des itérations (diversification "intelligente"), on peut s'inspirer des travaux sur le problème de Tournées de Véhicules [LS98]. On peut générer une tournée de grande taille que l'on scinde ensuite en tournées de taille acceptable.

Comment gérer les voisinages ?

Pour générer le voisinage d'un ensemble de tournées, on peut procéder de plusieurs manières :

- on peut appliquer un voisinage monotournée sur une ou plusieurs tournées choisies au hasard ;
- on peut aussi appliquer un voisinage spécifique multitournée.

On pourra choisir une des différentes manières de procéder de façon complètement aléatoire. De toute façon, il est utopique d'essayer de générer tout le voisinage pour n'en choisir qu'un seul élément. Il suffit de trouver un voisin améliorant pour la méthode GRASP et n'importe quel voisin réalisable est admissible pour la méthode Tabou.

La figure (3.20) montre un exemple de voisinage spécifique multitournée inspirés des heuristiques classiques développées pour les problèmes de tournées de véhicules [LS98]. On fait l'échange de deux points de contrôles qui n'appartiennent pas à la même géodésique : dans l'exemple ci-dessous, les points de contrôle repérés (A) et (3) .

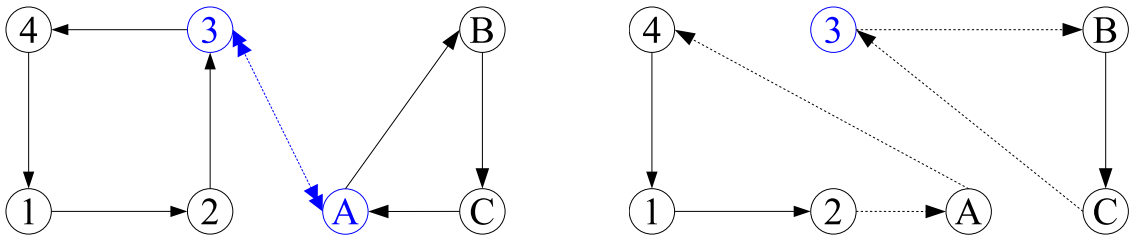


FIG. 3.20 – Échange de points de contrôle de géodésiques différentes

Maintenant, en ce qui concerne le Tabou, on maintenait une liste de nœuds qui ne pouvaient pas être points de contrôle pendant un certain nombre d'itérations. On peut gérer autant de listes qu'il y a de géodésiques, ou alors se contenter d'une liste globale. Cela peut permettre de créer un système de mise à l'index plus général (dans le sens multitournée).

3.6.3. Expérimentations

Nous n'avons pas fait d'expérimentations approfondies sur le problème multitournée. Nous nous sommes contentés de versions édulcorées des algorithmes sus-cités.

En ce qui concerne l'évaluation du trajet des produits, nous disposons du mode PCC, du mode réaliste avec lissage de chemin et d'un mode réaliste où aucun changement de ligne n'est autorisé.

La diversification est aléatoire. Les voisinages multitournées sont le choix au hasard d'un opérateur de voisinage monotournée sur une tournée choisie elle-aussi aléatoirement ou l'utilisation d'un voisinage multitournée présenté à la section précédente.

Même avec ces versions simplifiées, nous avons obtenu des résultats encourageants et nous avons réussi à améliorer le fonction objectif lorsque l"utilisation d'une seule géodésique ne permettait plus d'améliorer l'objectif (pour les valeurs relativement grandes de $p$max.

Prenons par exemple, l'instance *a10e*. On s'intéresse à la zone de $p_{\max}$ où l'utilisation d'une seule tournée ne permet pas de retrouver la solution optimale. On va essayer d'ajouter une deuxième tournée et mesurer son influence sur la fonction objectif :

- la résolution exacte donne $S = 28.37$ pour $p_{\max} = 200000$ avec le modèle m_1 ;
- la résolution monotournée donne $S = 23.28$ avec une 8-géodésique (147 itérations) ;
- la résolution multitournée donne $S = 25.07$ pour deux 4-géodésiques de taille maximale $p_{\max}$ chacune (202 itérations) ;

- la résolution multitournée donne $S = 23.16$ pour deux 4-géodésiques de taille maximale respective 150000 et 50000 (127 itérations).

Avec l'utilisation d'une tournée supplémentaire, il est donc possible d'améliorer l'objectif du monotournée et sur cette petite instance dans le même temps. On se rapproche donc de la solution optimale (on est passé de 18% à 11% de la solution optimale).

Voici un autre exemple sur l'instance *a20a*. Les différentes réplications sont résolues avec l'algorithme GRASP en mode réaliste.

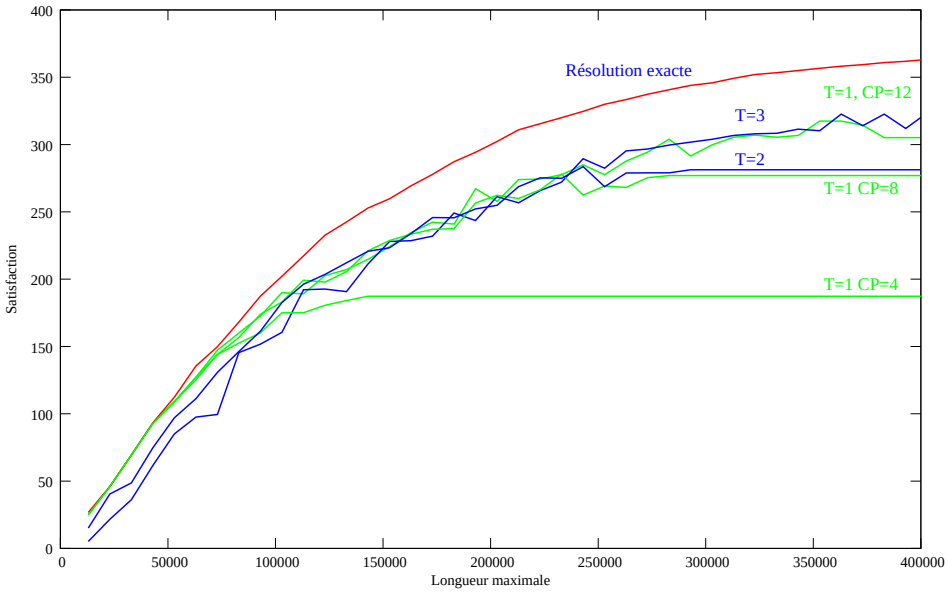


FIG. 3.21 – Résolution multitournée sur a20a

La figure ci-dessus (3.10) montre quelques exemples de résolution suivant le nombre de tournées. On a placé en abscisse le coût total généré par les tournées, en supposant qu'il n'y a pas de surcoût lié à l'utilisation d'une nouvelle tournée. Si l'on a considéré N tournées pour un coût global C, alors le coût de chaque tournée a été limité par $\frac{C}{N}$. Dans le cas multitournée, on s'est limité à l'usage de 4-géodésique.

Le graphe permet de comparer la résolution obtenue avec N 4-géodésiques et la résolution obtenue avec des tournées simples dont le nombre de points de contrôle est de l'ordre de $4.N$. Les courbes ne sont pas lissées, c'est-à-dire que l'on ne dispose pas de réplications obtenues en changeant la racine du générateur de nombres aléatoires. Ceci explique les vagues dans la mesure de la satisfaction. On peut affirmer que la résolution multitournée est pertinente dans la mesure où l'on supplante les résultats obtenus avec le monotournée dans des conditions similaires (mêmes limites sur le coût global et sur le nombre de points de contrôle). Ce mode de résolution est d'autant plus pertinent que la borne sur le coût est élevée. De plus, il n'y a pas de perte de temps liée à l'utilisation de l'algorithme multitournée.

Pour les faibles valeurs sur la borne de coût, le multitournée se comporte moins bien que le monotournée mais ceci est du à la distribution des contraintes sur les tournées (les tournées individuelles sont trop contraintes, trop petites pour satisfaire la même demande) et l'algorithme n'est pas capable d'ajuster la disposition des tournées pour retrouver la tournée plus grande équivalente.

3.7. Conclusion

Nous avons présenté deux adaptations de métaheuristiques pour les problèmes de synthèse de réseaux de mobilité avec demande élastique : la méthode GRASP et la méthode Tabou. Ces algorithmes utilisent une description particulière de circuit que l'on a baptisée géodésique : on ne manipule pas un circuit directement mais au travers de points particuliers surnommés points de contrôle qui sont liés entre eux par une certaine relation (un plus court chemin en général).

Les expérimentations menées, tant sur la version monotournée que sur la version multi-tournée, ont montré que l'on obtenait de "bonnes" solutions réalisables très rapidement. La qualité des solutions est calculée par la résolution de programmes entiers mixtes sur de petites instances et estimée par la résolution des problèmes relâchés pour les plus grands.

Nous sommes en mesure de retrouver les résultats obtenus grâce à la programmation linéaire et ce, bien plus rapidement pour les tournées simples dont la borne sur le coût n'est pas trop élevée (là où les temps de calcul de la résolution par PL sont les plus importants). Lorsque les bornes sur le coût sont plus élevée, il est nécessaire d'augmenter le nombre de tournées et le nombre de points de contrôle pour retrouver les solutions optimales.

On a donc pu vérifier la qualité des solutions obtenues avec les métaheuristiques et valider le comportement des adaptations des métaheuristiques et de la structure utilisée. L'emploi des modèles linéaires a permis de combler une lacune inhérente à ce type de résolution : le fait d'être incapable de se positionner par rapport aux solutions optimales. Les temps de résolution des métaheuristiques sont aussi bien meilleurs que les temps de résolution linéaire. Les heuristiques offrent aussi une plus grande souplesse et nous avons pu changer le type de fonction d'évaluation et considérer différents modes de routage pour les produits circulants sur le réseau.

Chapitre 4

Approche de Benders pour un modèle de Couplage de Flot Entier avec un Flot Fractionnaire

Sommaire

Dans les chapitres précédents, les chapitres 2 et 3, nous nous sommes intéressés aux problèmes de synthèse de réseaux de mobilité dont la caractéristique principale, outre le fait que la demande était élastique, était de mettre de côté la notion de temps. Le système modélisé de la sorte se trouvait dans un état stationnaire où les problème de connexion temps pouvaient être éludés. Nous voulons, dans ce chapitre, revenir sur cette hypothèse très forte et **proposer un modèle qui intègre la notion de temps**.

Nous allons tout d'abord donner une formulation très générale sous la forme d'un programme linéaire en nombres mixtes, proche des problèmes de type CFA (*Capacity Flow Assignment*) présentés au chapitre 1 — toujours exprimé en termes de flot entier couplé avec un multiflot fractionnaire — pour se concentrer sur une version orientée transport par la donnée

de triplets (origine, destination, demande). Cette approche particulière se justifie par le fait que l'on ne cherche pas un système de connexions fixes mais un objet circulant (un ensemble de véhicules) que les passagers ne sont pas obligés d'utiliser pour se déplacer.

Nous nous attaquerons ensuite au processus de résolution en utilisant une technique de synthèse de l'information comme l'agrégation du multiflot produit. Nous appliquerons tout d'abord la méthode de décomposition de Benders et nous proposerons un schéma heuristique de résolution inspiré par cette méthode. Nous terminerons ce chapitre par des expérimentations numériques.

4.1. Présentation des problèmes CFEMF et CFEMF-OD

4.1.1. Problème CFEMF

On veut résoudre le problème du Couplage d'un Flot Entier et d'un Multiflot Fractionnaire (CFEMF).
Pour un multiflot $f = (f^k)_{k \in K}$, on note $Sum(f) = \sum_{k \in K} f^k$ le flot agrégé.

Trouver sur un réseau $G = (V, E)$, un flot entier $F \geq 0$ et un multiflot $f = \{f^k, k \in K\} \geq 0$ qui **minimise** $c.F + p.Sum(f)$
sous les contraintes

$$F \leq MAX \tag{24.1}$$

$$C_{min} \leq f \leq C_{max} \tag{24.2}$$

$$\sum_{e \in \omega^-(v)} F_e - \sum_{e \in \omega^+(v)} F_e = 0 \qquad \forall v \in V \tag{24.3}$$

$$\sum_{e \in \omega^-(v)} f_e^k - \sum_{e \in \omega^+(v)} f_e^k = 0 \qquad \forall v \in V, \forall k \in K \tag{24.4}$$

$$Sum(f)_e \leq F_e \qquad \forall e \in A \tag{24.5}$$

$$F \in \mathbb{N} \tag{24.6}$$

$$f \in \mathbb{R}^+ \tag{24.7}$$

connaissant un sous-ensemble support $A \subset E$, un support d'indexation K, un vecteur capacité entier $MAX = (MAX_e)_{e \in E}$, deux familles de vecteurs $C^k_{min} = (C^k_{min_e})_{e \in E}$ et $C^k_{max} = (C^k_{max_e})_{e \in E}$, et des vecteurs coûts $c \geq 0$ et $p \geq 0$ indexés sur E.

Pl. 24 : Problème CFEMF

Les contraintes (24.3) et (24.4) sont les contraintes de conservation de flot pour F et f. Les contraintes (24.5) sont les contraintes de couplage entre le flot F et le multiflot f.

4.1.2. Problème CFEMF-OD

Ce problème est une spécialisation de CFEMF où chaque composante du multiflot doit router une certaine quantité entre une Origine et une Destination fixées. Comme nous l'avons

déjà évoqué, cette approche particulière se justifie par le fait que l'on ne cherche pas un système de connexions fixes mais un objet circulant que les passagers ne sont pas obligés d'utiliser pour se déplacer.

Contrairement aux formulations classiques, la contrainte de couplage ne concerne que certains arcs du réseau et il en résulte qu'il n'est plus possible d'envisager une approche polyédrale fondée sur la relaxation de la contrainte d'intégrité du flot : il n'est à priori plus possible d'introduire des coupes sur le flot traduisant le fait que tous les arcs supports de F doivent connecter tout couple de sommets origine-destination de l'ensemble de commodités. De plus, la contrainte d'intégrité sur F reflète le besoin, voire la nécessité, qu'ont les usagers de se regrouper (économie d'échelle) afin de partager un réseau d'infrastructures faiblement maillé par rapport au réseau initial. La relaxation de cette contrainte fournit de mauvais résultats, reflétant cette absence de partage, et ce, contrairement à ce qui est observable dans les applications en Télécommunications.

On normalise les demandes ($\sum_{k \in K} D^k = 1$) et on suppose que chacune des demandes est petite devant l'unité.

Description du problème

Le problème s'énonce ainsi :

Trouver dans un réseau $G = (V, E)$, un flot entier $F \geq 0$ et un multiflot $f \geq 0$ qui **minimise** $c.F + p.Sum(f)$
sous les contraintes

$$\sum_{e \in \omega^-(v)} F_e - \sum_{e \in \omega^+(v)} F_e = 0 \qquad \forall v \in V \tag{25.1}$$

$$\sum_{e \in \omega^-(v)} f_e^k - \sum_{e \in \omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \in K \tag{25.2}$$

$$Sum(f)_e \leq F_e \qquad \forall e \in A \tag{25.3}$$

$$0 \leq f^k \leq D^k \tag{25.4}$$

$$F \in \mathbb{N} \tag{25.5}$$

connaissant un sous-ensemble $A \subset E$, des vecteurs coûts $c \geq 0$ et $p \geq 0$ indexés sur E et une famille $OD = \{(o_k, d_k), k \in K\}$ de couples origine-destination et une famille $D = \{D_k, k \in K\}$ de demandes associées.

Pl. 25 : Problème CFEMF-OD

La variable b_v^k est définie comme suit :

$$\text{pour chaque commodité } k, \quad b_v^k = \begin{cases} -D^k & \text{si } v = o_k \\ +D^k & \text{si } v = d_k \\ 0 & \text{sinon} \end{cases}$$

Le problème CFEMF-OD est bien un problème CFEMF car pour se retrouver dans le cas du problème précédent, il suffit d'ajouter un arc fictif pour chaque commodité. Cet arc

ira de d^k à o^k, de capacités minimale et maximale identiques et égales à la demande D^k de la commodité. Tous les arcs du graphe auront une capacité minimale nulle et une capacité maximale infinie (ou supérieure à la demande maximale qu'ils peuvent satisfaire).

Taille du problème

Variables	
$\mathbb{N}$	m
$\mathbb{R}$	Km

Contraintes	
égalités	$(K+1)n$
"inférieur ou égal"	$a + 2m(K+1)$
	$a + (2m+n)(K+1)$

Le problème exprimé en termes de programme linéaire mixte est rendu difficile par la présence de m variables entières (une pour chaque arc du graphe).

4.1.3. Exemple d'application : modélisation d'un problème de transport

Il s'agit de modéliser un problème de transport du même ordre que ceux que l'on a présentés au chapitre 2, en introduisant **la notion de temps**. On devra, par exemple, respecter une nouvelle contrainte comme la **date au plus tard d'arrivée à destination**.

Cette nouvelle contrainte est un élément constituant majeur pour une commodité. Prendre en compte une telle contrainte pourra faire exploser le nombre de commodités. En effet, pour un même trajet origine-destination, si l'on dispose de plusieurs dates d'arrivée au plus tard, on définira autant de commodités.

On va faire l'optimisation sur un horizon de temps prédéterminé et on crée un **graphe temporel discrétisé** afin d'essayer d'en limiter la complexité. Malheureusement, cette manière de faire pourra introduire un biais de résolution.

On considérera deux modes de transport : un mode de transport rapide, en conteneur et un mode de déplacement lent lorsque le produit se déplace hors d'un conteneur.

Soit un graphe $G = (V, E)$ représentatif d'un réseau de transport. On a $E = A \cup \overline{A}$ où A est l'ensemble des arcs de type "rapide" et $\overline{A}$ l'ensemble des arcs de type "lent".

Soit un ensemble de commodités définies par le quintuplet $(o^k, d^k, D^k, t^k, T^k)$ où o^k est l'origine, d^k est la destination, D^k est la demande, t^k est la date pour laquelle le trajet doit être réalisé, T^k est la durée maximale acceptable du trajet.

Les conteneurs partent tous d'un dépôt unique noté D et sont identiques de capacité donnée.

On choisit un pas de temps δ et un entier N tels que tous les produits soient acheminables de leur origine à leur destination sur une échelle de temps allant de 0 à $N\delta$.

On va construire un graphe dynamique $G^* = (V^*, E^*)$ prenant en compte certaines contraintes temporelles, les différents modes de transport possibles et la discrétisation temporelle.

Pour chaque sommet $v \in V$, on crée $N+1$ copies étiquetées $v_0, \ldots, v_N$. On repérera donc facilement dans G^* le sommet original de G par la mise en indice de l'instant correspondant.

Pour établir les lois de conservation de flot, on introduit deux sommets fictifs U et D pour représenter respectivement les produits et le dépôt (par commodité, on garde le même nom que le sommet de G).

Ainsi $V^* = \{v_r, v \in V, r = 0 \ldots N\} \cup \{U, D\}$

Si l'on note $\ell(e)$, le temps de parcours de l'arc e, on détermine les quantités suivantes : $\ell^*(e) = \lceil \frac{\ell(e)}{\delta} \rceil$, $t^{*k} = \lceil \frac{t^k}{\delta} \rceil$

Tous les arcs suivants définissent E^* :

- deux arcs $[v_r, v_{r+1}]$ pour $v \in V$ et $r : 0 \dots N-1$, l'un de type rapide et l'autre de type lent ;
- $[D, D_r]$ et $[D_r, D]$ pour $r : 0 \dots N$, arcs de type rapide ;
- $[U, o_r^k]$ et $[d_r^k, U]$ pour tout k et $r : 0 \dots N$, arcs de type lent ;
- $[x_r, y_{r+\ell^*(e)}]$ pour tout $e = [x, y] \in A$ et $0 \leq r \leq N - \ell^*(e)$, arcs de type rapide ;
- $[x_r, y_{r+\ell^*(e)}]$ pour tout $e = [x, y] \in \overline{A}$ et $0 \leq r \leq N - \ell^*(e)$, arcs de type lent.

Tous les arcs lents définissent $\overline{A}^*$ et tous les arcs rapides définissent A^*. On a bien entendu $A^* \cup \overline{A}^* \subset E^*$.

Le problème de transport prend alors la forme d'un problème CFEMF-OD de la forme :

On cherche dans le réseau G^* défini précédemment, un flot conteneur (véhicule) F et le multiflot produit f tels que :

- $cF + pSum(f)$ est minimale ;
- $F_e = 0$ pour tout $E \in \overline{A}^*$;
- $f_e^k = D^k$ pour $e = [U, o_r^k]$ où $r = \lceil \frac{t^k - T^k}{\delta} \rceil$ (cf Note 1) ;
- $f_e^k = 0$ pour tout arc $e = [x, y_r]$ tel que $r > t^{*k}$;
- $Sum(f)_e \leq F_e$ pour chaque $e \in A^*$ (on peut introduire ici la capacité des conteneurs).

Pl. 26 : Interprétation d'un problème de type CFEMF-OD

Note 1 : Dans ce cas, on considère que les passagers partent le plus tard possible, sinon il faut plutôt utiliser la contrainte suivante :

$$\sum_{e=[x,o_r^k] | r \leq \lceil (t^k - T^k)/\delta \rceil} f_e^k = D^k$$

On cherche à minimiser la quantité $\sum_{e \in A^*} L_e F_e$ où

$$L_e = \begin{cases} \ell_e & \text{si } e \text{ est de la forme } [x_r, y_{r+\ell_e^*}] \\ \mu & \text{si } e \text{ est de la forme } [x_r, x_{r+1}] \\ \alpha & \text{si } e \text{ est de la forme } [D, D_r] \\ 0 & \text{ailleurs} \end{cases}$$

avec μ et α deux constantes données représentant des coûts liés à la chronologie (délai, temps d'attente, retard, etc).

4.2. Méthode de résolution exacte : décomposition de Benders

Nous allons appliquer simultanément le principe de décomposition de Benders aux problèmes CFEMF et CFEMEF-OD. Ces problèmes définissent deux types de variables : un flot entier et un multiflot fractionnaire. Une approche classique serait d'orienter la résolution autour du flot entier. Cependant, en remarquant que les problèmes réécrits avec le multiflot f fixé (respectivement CFEMF$_f$ et CFEMF-OD$_f$) sont de simples problèmes de recherche de flot entier de coût minimum (flot avec capacités entières mais pas forcément avec des coûts entiers) et que le multiflot agrégé $Sum(f)$ joue un rôle particulier dans ces problèmes, nous allons choisir ce dernier comme élément central de la décomposition.

Trouver sur un réseau G, un flot F qui **minimise** $c.F + p.Sum(f)$ (le multiflot f est fixé)
sous les contraintes

$$F_e \leq MAX_e \qquad \forall e \in A \tag{27.1}$$
$$Sum(f)_e \leq F_e \qquad \forall e \in A \tag{27.2}$$
$$\sum_{e \in \omega^-(v)} F_e - \sum_{e \in \omega^+(v)} F_e = 0 \qquad \forall v \in V \tag{27.3}$$
$$F \in \mathbb{N} \tag{27.4}$$

Pl. 27 : Problème CFEMF$_f$

Trouver sur un réseau G, un flot F qui **minimise** $c.F + p.Sum(f)$
sous les contraintes

$$Sum(f)_e \leq F_e \qquad \forall e \in A \tag{28.1}$$
$$\sum_{e \in \omega^-(v)} F_e - \sum_{e \in \omega^+(v)} F_e = 0 \qquad \forall v \in V \tag{28.2}$$
$$F \in \mathbb{N} \tag{28.3}$$

Pl. 28 : Problème CFEMF-OD$_f$

Le flot F est entier. Ainsi, remplacer respectivement les contraintes (27.2) et (28.1) par les contraintes (27.2a) et (28.1a) ne change en rien les problèmes initiaux :

$$\lceil Sum(f)_e \rceil \leq F_e \qquad \forall e \in A \tag{27.2a}$$
$$\tag{28.1a}$$

Exprimons maintenant les duaux des relaxations continues de ces problèmes (ces expressions seront utilisées ultérieurement). Nous reportons dans le dual la constante en f. On associe les valeurs duales α, $\lambda = (\lambda_e)_{e \in A}$ et μ respectivement aux contraintes (27.1), (27.2) et (27.3). On peut ainsi donner l'expression du problème dual DU_f de CFEMF_f, et $\mathrm{DU\text{-}OD}_f$ de $\mathrm{CFEMF\text{-}OD}_f$.

Trouver α, λ et μ telles que $\lceil Sum(f) \rceil . \lambda - MAX.\alpha + p.Sum(f)$ soit **maximale** sous les contraintes

$$-\alpha_e + \lambda_e + \mu_x - \mu_y \leq c_e \qquad \forall e = (x, y) \in E \tag{29.1}$$
$$\alpha \geq 0, \lambda \geq 0, \mu \in \mathbb{R} \tag{29.2}$$

Pl. 29 : Problème DU_f

Trouver λ et μ telles que $\lceil Sum(f) \rceil . \lambda + p.Sum(f)$ soit **maximale** sous les contraintes

$$\lambda_e + \mu_x - \mu_y \leq c_e \qquad \forall e = (x, y) \in E \tag{30.1}$$
$$\lambda \geq 0, \mu \in \mathbb{R} \tag{30.2}$$

Pl. 30 : Problème $\mathrm{DU\text{-}OD}_f$

En ce qui concerne les contraintes (29.1) et (30.1), on a étendu la définition de λ sur E et on a donc posé : $\lambda_e = 0, \quad \forall e \notin A$.

Le principe de la décomposition de Benders dans le cas général est repris à la section 1.2.1 (page 10). Rappelons toutefois quelques définitions sur les objets manipulés.

On appelle **programme maître** (PM), le programme qui contient toutes les coupes nécessaires à la résolution optimale. On note I^* cet ensemble de coupes. On appelle **programme maître restreint** (PMR) un programme semblable au programme maître mais qui ne contient qu'un sous-ensemble de ces contraintes I^*.

Résoudre le PM, c'est résoudre le problème initial.

Dans la suite de ce document, pour plus de simplicité, nous allons utiliser exclusivement les notations introduites pour la résolution de CFEMF-OD.

Rechercher Z et f telles que Z soit **minimale**
sous les contraintes

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \in K \tag{31.1}$$

$$Z - p.Sum(f) - \lambda^i \lceil Sum(f) \rceil \geq 0 \qquad \forall i \in I^* \tag{31.2}$$

Pl. 31 : Programme Maître (PM) issu de CFEMF-OD

Ce problème pour être résolu fait appel à un schéma de décomposition maître-esclave, qui résout alternativement un problème maître restreint et un problème esclave.

Rechercher Z et f telles que Z soit **minimale**
sous les contraintes

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \in K \tag{32.1}$$

$$Z - p.Sum(f) - \lambda^i \lceil Sum(f) \rceil \geq 0 \qquad \forall i \in I \subset I^* \tag{32.2}$$

Pl. 32 : Programme Maître Restreint (PMR) issu de CFEMF-OD

Les contraintes (32.2) sont des coupes de Benders, dites de type I, remontées du programme esclave vers le programme maître. Pour chaque solution réalisable du problème maître restreint, on obtient une borne inférieure sur la solution optimale (il s'agit d'une borne inférieure dans la mesure où il manque des contraintes au problème final).

Les problèmes esclave (PE) sont de la forme :

Trouver le flot F qui **minimise** $c.F$
sous les contraintes

$$\lceil Sum(f)_e \rceil \leq F_e \qquad \forall e \in A \tag{33.1}$$

$$\sum_{e\in\omega^-(v)} F_e - \sum_{e\in\omega^+(v)} F_e = 0 \qquad \forall v \in V \tag{33.2}$$

$$F \in \mathbb{R} \tag{33.3}$$

Pl. 33 : Problème Esclave issu de CFEMF-OD

Remarquons que le programme esclave que l'on cherche à minimiser doit respecter une borne inférieure mais pas de borne supérieure. Ainsi le problème est **toujours réalisable**, son **dual** — calculé précédemment — est obligatoirement **borné** et, par conséquent, le schéma

de décomposition que nous donnons est bien un schéma de Benders (même si on résout le primal alors que l'algorithme de Benders favorise le dual). Il n'est pas nécessaire d'introduire le deuxième type de contrainte qui intervient uniquement lorsque le problème esclave n'est pas borné. Pour chaque solution réalisable du problème esclave, on obtient une borne supérieure sur la solution finale. Comme nous l'avons déjà fait remarquer, le PE est un problème facile à résoudre.

D'après [GG74], on peut se contenter de résoudre le programme maître restreint de manière sous-optimale. Il suffit de déterminer une solution réalisable "suffisamment intéressante". Cette variante est motivée par la mauvaise qualité - due au manque d'informations (de coupes) sur le programme maître - des coupes générées dans les premières itérations.

La résolution du PMR doit normalement donner une borne inférieure. Bien entendu, si ce PMR n'est pas résolu à l'optimum, on ne peut plus disposer d'un encadrement sur la solution optimale et donc mesurer la qualité de la solution courante.

4.3. Méthode de résolution heuristique

Nous allons nous inspirer du schéma de décomposition de Benders pour proposer un schéma heuristique de résolution, lui-aussi de type maître-esclave.

On suppose tout d'abord qu'il existe une solution réalisable (F_0, f_0) du problème (que ce soit CFEMF ou CFEMF-OD). Le flot F_0 est choisi tel qu'il soit solution optimale du problème restreint à f_0 donné.

Si l'on reprend l'expression du dual du problème restreint (PL 29 et PL 30), on remarque que trouver un couple améliorant (F, f) pour le problème choisi, c'est modifier f tout en respectant les contraintes de positivité et de capacité et tel que la quantité $\lambda.\lceil Sum(f)\rceil + p.Sum(f) - \alpha.MAX$ diminue (la fonction-objectif du dual).

La réciproque est bien évidemment fausse, on est donc amené à introduire l'un des problèmes auxiliaires $P_{\text{aux}}(\lambda)$ suivants :

Trouver le multiflot f qui **minimise** $\lambda.\lceil Sum(f)\rceil + p.Sum(f)$
sous les contraintes

$$C_{min} \leq f \leq C_{max} \tag{34.1}$$

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = 0 \qquad \forall v \in V, \forall k \in K \tag{34.2}$$

$$f \in \mathbb{R}^+ \tag{34.3}$$

Pl. 34 : Problème $P_{\text{aux}}(\lambda)$ pour CFEMF

ou encore

Trouver le multiflot f qui **minimise** $\lambda.\lceil Sum(f)\rceil + p.Sum(f)$
sous les contraintes

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \in K \tag{35.1}$$

$$f \in \mathbb{R}^+ \tag{35.2}$$

Pl. 35 : Problème $P_{\text{aux}}(\lambda)$ pour CFEMF-OD

Ce problème dans le cas général est un problème difficile : c'est une instance du problème de Multiflot Fractionnaire de Coût Minimal (MFCM, voir section 5.2) mais nous verrons que dans certains cas (multiflot réduit à un flot simple ou multiflot dont les composantes sont indépendantes), il peut se résumer à un problème plus facile à résoudre de type FCEM. Ce problème est aussi une instance du PMR où l'on ne considère qu'une seule coupe.

Pour valider le schéma de décomposition, nous allons utiliser un solveur : P_{aux} va donc être réinterprété en terme de multiflot fractionnaire dominé par un flot entier. Cette transformation va donc à l'encontre de tous les efforts prodigués jusqu'à maintenant mais elle reste nécessaire pour la validation. Nous essaierons de résoudre ce problème auxiliaire avec une heuristique dans la mesure où il utilise le flot agrégé $Sum(f)$ qui synthétise la façon dont les usagers mutualisent leur stratégie de déplacement et que les heuristiques permettent de traiter cet objet explicitement sous la forme de chemins ou de circuits.

Voici finalement le schéma de résolution par décomposition maître-esclave que nous proposons :

Algorithme 12 : Schéma heuristique DME

```
Initialiser F et f réalisables pour CFEMF
Stop ← faux
tant que non (Stop) faire
    Résoudre CFEMF_f en relâchant la contrainte d'intégrité de F
    Extraire la composante duale λ* associée
    Évaluer Z = c.F + pSum(f), Calculer λ = Λ(λ, λ*)
    Améliorer P_aux(λ)
    si f est inchangé OU Z n'a pas été amélioré depuis un certain temps alors
        Stop ← vrai
    fin si
fin tant que
```

Ce schéma est heuristique car on ne maintient pas un pool complet de coupes. On ne garde que des valeurs significatives des valeurs duales. L'un des problèmes de ce type d'approche est justement de déterminer un bon jeu de valeurs duales. Ce "bon" jeu est dissimulé derrière la fonction Λ qui est fonction du jeu de paramètres obtenu à l'itération précédente et en fonction de nouvelles informations permet d'obtenir un vecteur que l'on injecte dans la résolution de P_{aux}.

Remarquons aussi que l'on ne cherche pas vraiment à résoudre P_{aux} si cela est trop difficile mais seulement à l'améliorer.

4.4. Expérimentations numériques

Dans ce paragraphe, nous allons donner le fruit de quelques expérimentations concernant les méthodes suivantes :

- la résolution directe par un solveur
- la résolution par la méthode de Benders classique, en réinjectant toutes les coupes obtenues dans le pool des coupes

Le PMR reste un problème dur à résoudre dans la mesure où c'est un programme en nombres mixtes, on est donc très vite limité par la taille des instances.

A la section 5.2, nous décrivons le problème du multiflot entier à coût minimal, ce problème est un cas particulier du PMR où l'ensemble des contraintes est réduit à un seul élément. Nous voulons déterminer si, dans certains cas, il est possible de résoudre un problème de type MFCM en lieu et place d'un PMR, et ainsi utiliser le schéma de décomposition heuristique décrit à la section précédente. Le problème MFCM est paramétré par un vecteur λ, alors que le PMR, lui, est défini par un ensemble de vecteurs λ_i grossissant avec le nombre d'itérations requises pour la résolution. L'idée sous-jacente est de trouver un moyen de synthétiser l'information portée par le pool de paramètres $(\lambda_i)_{i\geq 0}$ dans un seul vecteur λ pour proposer une heuristique de résolution (fonction que l'on a noté Λ). Nous allons envisager trois cas de figure :

- $\Lambda(\lambda, \lambda^*) = \lambda^*$ (ou encore "$\lambda = \lambda^i$"), c'est-à-dire un vecteur paramètre égal aux valeurs duales que l'on vient de trouver à la dernière itération. On ne prend donc en considération que la dernière coupe générée, en oubliant toutes les précédentes.
- $\Lambda(\lambda, \lambda^*) = (\max(\lambda_e, \lambda_e^*))_{e\in E}$ (ou encore $\lambda = \max \lambda^i$), on exploite les informations des coupes en ne gardant que les composantes les plus importantes.
- $\Lambda_I(\lambda, \lambda^*) = \dfrac{I\lambda + \lambda^*}{I+1}$ (ou encore $\lambda = \dfrac{\sum_i \lambda^i}{I}$), on essaie de lisser les différentes informations de coupes. C'est ce schéma par son aspect synthétique qui semble le plus prometteur bien qu'il puisse accentuer le défaut principal de Benders, c'est-à-dire la lenteur de convergence vers la solution optimale.

Les instances de test que nous avons utilisées ont pour support les supports des réseaux déjà définis dans ce manuscrit, c'est-à-dire les graphes à 10 et à 20 sommets, précédemment utilisés. La taille est déjà respectable au vu de la complexité spatiale déjà rencontrée (rappelons que le nombre de variables entières est égal au nombre d'arcs de A). Les coûts mis en jeu pour le multiflot sont les distances euclidiennes sur le graphe, les coûts sur le flot sont un ratio sur les distances euclidiennes ($c_e = r.p_e, \forall e \in E$). Le temps de calcul d'un PL a été arbitrairement limité à 6h.

Pour un ratio r nul, en évinçant le coût du flot F, on s'attend à ce que les commodités soient routées indépendamment et au meilleur coût (relativement au vecteur p). Si au contraire, ce ratio est grand, le coût de F est trop élevé et le routage des commodités doit se faire sur des arcs n'appartenant pas à A (l'ensemble sur lequel F et f sont couplés). Entre ces deux valeurs extrêmes, on s'attend donc à ce que les transports soient mutualisés.

En ce qui concerne la résolution avec solveur, on donne en colonnes : le nom de l'instance résolue, le ratio de coût. On relève après résolution, la valeur de la fonction-objectif, si elle

est optimale (O) ou non, le temps de calcul, le nombre d'itérations (le nombre de simplexes résolus), on note aussi si le multiflot n'a pas été routé suivant des chemins (É).

instance	ratio	Z	O	temps	itérations	É
a10e	0	25612.8	√	<1s	138	
a10e	0.033	49135.6	√	25s	51213	
a10e	0.33	77614.5	√	3s	1799	
a10e	1	77614.5	√	<1s	264	
a10e	10	77614.5	√	<1s	241	
a10a	0	20859.4	√	<1s	364	
a10a	0.033	44853.8	√	1h	2426482	
a10a	0.33	63210.4	√	8s	5394	
a10a	1	63210.4	√	<1s	528	
a10a	10	63210.4	√	<1s	512	
a20e	0	25173.4	√	<1s	662	
a20e	0.033	53665.1		5h20	5598467	
a20e	0.33	76282.1	√	192s	30718	
a20e	1	76282.1	√	<1s	954	
a20e	10	76282.1	√	<1s	928	
a20a	0	19822.8	√	8s	2908	
a20a	0.033	49939.9		5h20	1604894	
a20a	0.33	60069.1	√	2081s	104995	
a20a	1	60069.1	√	3s	3234	
a20a	10	60069.1	√	3s	3234	

Tab. 4.1 – Résultats avec CPLEX

Nous avons pu obtenir des résultats assez rapidement pour toutes les instances traitées. Nous remarquons que les instances avec un faible ratio (r=0.033) sont très difficiles à calculer en comparaison des autres ratios. CPLEX offre différentes approches de résolution pour le programme en nombres mixtes. Nous avons laissé les paramètres par défaut, c'est-à-dire une résolution par le dual. En fait, on vérifie que dès que $r > 0.33$, le coût de F est trop élevé et F n'est pas utilisé (c'est pour cela que les temps de calculs sont courts et les coûts identiques : le routage du multiflot f se fait suivant les arcs de $\overline{A}$).

instance	ratio	Z	O	temps(s)	itérations	S
a10e	0	25612.8	√	<1s	1	
a10e	0.033	49255.0		10h	53	
a10e	0.33	77614.5	√	11s	3	
a10e	1	77614.5	√	<1s	3	
a10e	10	77614.5	√	<1s	3	
a10a	0	20859.4	√	<1	1	
a10a	0.033	44853.8	√	26h	5	
a10a	0.33	63210.4	√	90s	2	
a10a	1	63210.4	√	4s	2	
a10a	10	63210.4	√	5s	2	

Tab. 4.2 – Résultats obtenus par la méthode de Benders

Avec la méthode de Benders, on retrouve les mêmes résultats qu'avec la résolution directe par CPLEX. On note toutefois que les temps de calculs sont sensiblement plus longs. En effet,

avec la méthode directe (éventuellement limitée en temps), un seul PL est résolu. La méthode de Benders est itérative et à chaque itération, un nouveau PL doit être résolu (le PMR), ce qui explique cette explosion des temps de calculs. On donne le nombre d'itérations qui ont été nécessaires, ce nombre est égal au nombre de PL résolus mais aussi au nombre de coupes générées.

Sur les instances à 10 sommets, une seule n'est pas résolue à l'optimum, la convergence est trop lente et le processus s'arrête.

Sur ces instances, le nombre de coupes de Benders est relativement limité, il n'est donc pas vraiment nécessaire d'oublier les coupes qui ne sont plus utiles depuis un certain nombre d'itérations.

Nous donnons ensuite les expérimentations avec le schéma heuristique et différents jeux pour le coefficient λ. Les conditions d'arrêts de ces schémas sont : la valeur du paramètre λ est inchangée, un nombre maximal d'itérations a été effectué, un nombre d'itérations sans amélioration de l'objectif a été rencontré ou encore le fait que le multiflot soit inchangé après résolution du problème maître restreint ou de P_{aux}.

instance	ratio	Z	O	temps(s)	itérations	S
a10e	0	25612.8	√	<1s	2	
a10e	0.033	52870.7		50s	5	
a10e	0.33	89208.3		9s	3	
a10e	1	89208.3		<1s	3	
a10e	10	89208.3		<1s	3	
a10a	0	20859.4	√	1s	2	
a10a	0.033	47096.4		4s	2	
a10a	0.33	193201.0		13s	2	
a10a	1	520606.0		16s	2	
a10a	10	4.90217e+06		17s	2	

TAB. 4.3 – Résultats obtenus avec $\lambda = \max \lambda_i$

Même sur de petites instances, le jeu de poids défini avec l'opérateur *max* ne semble pas donner des résultats.

instance	ratio	Z	O	temps(s)	itérations	S
a10e	0	25612.8	√	<1s	2	
a10e	0.033	52382.3		44s	10	
a10e	0.33	105685		2h51	54	
a10e	1	89208.3		5s	3	
a10e	10	89208.3		<1s	3	
a10a	0	20859.4	√	1s	2	
a10a	0.033	47096.4		90s	51	
a10a	0.33	193201		109s	15	
a10a	1	520606		430s	54	
a10a	10	4.90217e+06		271s	35	

TAB. 4.4 – Résultats obtenus avec $\lambda = \text{moy}\lambda_i$

On peut faire le même constat que pour le jeu de poids précédent : celui-ci ne semble pas donner de résultats intéressants. Mais on peut noter que le comportement de ces méthodes est similaire, même si on est parfois très éloignés des objectifs calculés directement ou avec la décomposition de Benders.

Ces mauvais résultats sont peut-être dus à une optimisation trop brutale du programme maître, dans la mesure où le multiflot pour l'itération suivante du processus de résolution résulte de l'optimisation complète du programme maître. Il serait peut-être plus judicieux d'utiliser un multiflot résultant d'une optimisation moins poussée (c'est-à-dire un multiflot se trouvant dans la direction "optimale" mais non optimal).

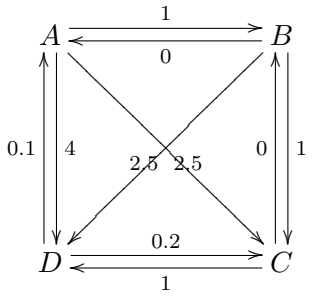
instance	ratio	Z	O	temps(s)	itérations	S
a10e	0	25612.8	√	<1s	1	
a10e	0.033	49255		29s	53	
a10e	0.33	117126		40s	53	
a10e	1	216592		5s	53	
a10e	10	1.5527e+06		5s	53	
a10a	0	20859.4	√	<1s	1	
a10a	0.033	44853.8	√	4h30	55	
a10a	0.33	63210.4	√	598s	53	
a10a	1	63210.4	√	17s	53	
a10a	10	63210.4	√	17s	53	

TAB. 4.5 – Résultats obtenus avec $\lambda = \lambda_i$

Les résultats concernant cette méthode sont partagés : cette méthode semble bien se comporter sur l'instance *a10a*, mais ce n'est pas le cas pour l'instance *a10e*. Nous ne savons pas si ce phénomène est du à la distribution ou au nombre de commodités.

En ce qui concerne l'instance *a10e*, on peut observer un phénomène de bascule. L'algorithme oscille entre deux valeurs qui peuvent même être éloignées de l'optimum. On peut essayer de gommer cet effet en paramétrant la qualité de la solution du programme maître : se déplacer, en quelque sorte, d'un certain pas dans la direction de l'optimisation sans pour autant prendre la valeur optimale

4.4.1. Exemple sur un petit graphe



Les commodités sont au nombre de deux :
- la première de A vers C de demande 0.4
- la seconde de B vers D de demande 0.6

Caractéristiques :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
c_e	1	0	1	0	1	0.2	0.1	4	2.5	2.5
p_e	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Le multiflot réalisable de départ est déterminé avec $\lambda_e = 0, \forall e$. On appelle λ^i les valeurs duales obtenues lors de la résolution du problème esclave à l'itération i. On appelle $\overline{\lambda}^i$ la valeur effectivement utilisée par le PMR à cette même itération.

Cas 1 : décomposition de Benders

Itération 1 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	0	0	0	1	0	0	1	0	1	1
λ_e	0	0	0	0	0	0	0	0	2.5	2.6

$Z = 5.6$ Z_λ de 5.6 à 1

Itération 2 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0.6	0	0	0	0	0	0.6	0	0
F_e	1	1	1	1	0	0	1	1	0	0
λ_e	1	0	1	0	0	0	0	4.1	0	0

$Z = 7.1$ Z_λ de 7.1 à 3

Itération 3 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	1	0	1	0	1	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	0

$Z = 4.1$ (valeur optimale) Z_λ de 4.1 à 2.9

Itération 4 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	0	0	1	1	1	0	1	0	1	0
λ_e	0	0	1	0	1.1	0	0	0	2.5	0

$Z = 5.4$ Z_λ de 5.4 à 1.7

Itération 5 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	1	0	1	1	0	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	2.6

$Z = 5.3$ Z_λ de 5.3 à 3

Itération 6 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	1	0	1	0	1	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	0

$Z = 4.1$ (valeur optimale) Z_λ inchangé (condition d'arrêt)

Cas 2 : $\overline{\lambda}^i = \lambda^i$

On ne tient compte que des dernières valeurs duales : $\overline{\lambda}^i = \lambda^i$

Itération 1 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	0	0	0	1	0	0	1	0	1	1
λ_e	0	0	0	0	0	0	0	0	2.5	2.6

$Z = 5.6$ Z_λ de 5.6 à 1

Itération 2 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0.6	0	0	0	0	0	0.6	0	0
F_e	1	1	1	1	0	0	1	1	0	0
λ_e	1	0	1	0	0	0	0	4.1	0	0

$Z = 7.1$ Z_λ de 7.1 à 0.5

On constate ensuite que l'itération 3, c'est l'itération 1 ; l'itération 4, l'itération 2, etc. On boucle donc à l'infini.

Cas 3 : $\overline{\lambda}_e^i = \max_{j \leq i} \lambda^i$

On prend $\overline{\lambda}_e^i = \max_{j \leq i} \lambda^i$.

Itération 1 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	0	0	0	1	0	0	1	0	1	1
λ_e	0	0	0	0	0	0	0	0	2.5	2.6
$\overline{\lambda}_e$	0	0	0	0	0	0	0	0	2.5	2.6

$Z = 5.6$ $Z_{\overline{\lambda}}$ de 5.6 à 1

Itération 2 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0.6	0	0	0	0	0	0.6	0	0
F_e	1	1	1	1	0	0	1	1	0	0
λ_e	1	0	1	0	0	0	0	4.1	0	0
$\overline{\lambda}_e$	1	0	1	0	0	0	0	4.1	2.5	2.6

$Z = 7.1$ $Z_{\overline{\lambda}}$ de 7.1 à 3

Itération 3 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	1	0	1	0	1	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	0
$\overline{\lambda}_e$	1	0	1	0	1.1	0	0	4.1	2.5	2.6

$Z = 4.1$ (valeur optimale) $Z_{\overline{\lambda}}$ inchangé (condition d'arrêt)

Cas 4 : $\overline{\lambda}_e^i = \dfrac{\sum_{j \leq i} \lambda^i}{i}$

On prend $\overline{\lambda}_e^i = \dfrac{\sum_{j \leq i} \lambda^i}{i}$.

Itération 1 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	0	0	0	1	0	0	1	0	1	1
λ_e	0	0	0	0	0	0	0	0	2.5	2.6
$\overline{\lambda}_e$	0	0	0	0	0	0	0	0	2.5	2.6

$Z = 5.6$ $Z_{\overline{\lambda}}$ de 5.6 à 1

Itération 2 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0.6	0	0	0	0	0	0.6	0	0
F_e	1	1	1	1	0	0	1	1	0	0
λ_e	1	0	1	0	0	0	0	4.1	0	0
$2\overline{\lambda}_e$	1	0	1	0	0	0	0	4.1	2.5	2.6

$Z = 7.1$ $Z_{\overline{\lambda}}$ de 7.1 à 3

Itération 3 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	1	0	1	0	1	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	0
$3\overline{\lambda}_e$	2	0	2	0	1.1	0	0	4.1	2.5	2.6

$Z = 4.1$ (valeur optimale) $Z_{\overline{\lambda}}$ de 6.1 à 5.6

Itération 4 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0	0	0	0.4	0
f_e^2	0	0	0	0	0	0	0	0	0	0.6
F_e	0	0	0	1	0	0	1	0	1	1
λ_e	0	0	0	0	0	0	0	0	2.5	2.6
$4\overline{\lambda}_e$	2	0	2	0	1.1	0	0	4.1	5	5.2

$Z = 5.6$ $Z_{\overline{\lambda}}$ de 10.7 à 5.1

Itération 5 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0	0	0	0	0	0.4	0	0.4	0	0
f_e^2	0	0.6	0	0	0	0	0	0.6	0	0
F_e	0	1	0	1	0	1	0	1	0	0
λ_e	0	0	0	0	0	0.2	0	4	0	0
$5\overline{\lambda}_e$	2	0	2	0	1.1	0.2	0	8.1	5	5.2

$Z = 5.2$ $Z_{\overline{\lambda}}$ de 9.3 à 6.1

Itération 6 :

	$[A,B]$	$[B,A]$	$[B,C]$	$[C,B]$	$[C,D]$	$[D,C]$	$[D,A]$	$[A,D]$	$[A,C]$	$[B,D]$
f_e^1	0.4	0	0.4	0	0	0	0	0	0	0
f_e^2	0	0	0.6	0	0.6	0	0	0	0	0
F_e	1	0	1	0	1	0	1	0	0	0
λ_e	1	0	1	0	1.1	0	0	0	0	0
$6\overline{\lambda}_e$	3	0	3	0	2.2	0.2	0	8.1	5	5.2

$Z = 4.1$ (valeur optimale) $Z_{\overline{\lambda}}$ de 9.2 à 10.7 (condition d'arrêt)

4.4.2. Vérification des hypothèses d'optimisation trop brutale

Le mauvais comportement de la méthode de résolution lorsque l'information des coupes est synthétisée peut-elle venir d'une optimisation trop forte ? En d'autres termes, il est nécessaire de vérifier que l'optimisation dans une direction donnée, une direction déduite des itérations précédentes, ne biaise pas le résultat final. Pour cela, on se propose de tester les différents cas possibles en prenant comme solution courante, non pas la solution optimale, mais une solution intermédiaire entre les deux solutions.

On appelle f^i la solution courante, f'^i la solution optimale calculée à partir de f^i, et $\eta \in]0,1]$ le pas d'incrément, alors on a naturellement :

$$f^{i+1} = f^i + \eta(f'^i - f^i)$$

Avec ces notations, on peut donc remarquer que l'on ne prend plus comme multiflot à l'itération suivante, le multiflot optimal f'^i mais bien $f^{i+1} = f^i + \eta(f'^i - f^i)$ où η est le pas d'incrément dans la direction d'optimisation.

Nous ne donnons pas de résultats, mais malheureusement, quelque soit le pas η envisagé (de 0.001 à 1), nous n'avons pas réussi à améliorer les résultats.

4.4.3. Amélioration du schéma heuristique

On peut donc déduire de tout cela qu'il est nécessaire de garder toutes les informations des différentes coupes, on propose donc de résoudre le problème de la manière suivante (on suppose que l'on sait résoudre ou trouver une bonne solution au problème MFCM).

Chaque coupe i peut s'exprimer comme $Z \geq Z_i(f)$ et on cherche à trouver un f tel que $Z = \max_i Z_i(f)$ soit minimal.

Il suffit d'isoler une coupe i serrée, c'est-à-dire que $Z = Z_i(f)$ et optimiser suivant cette coupe (on résout alors un problème de type MFCM). Après optimisation, on évalue à nouveau l'ensemble des coupes, la valeur de Z a pu alors complètement changer, mais on réitère le processus jusqu'à ce que l'on ne puisse plus trouver de multiflot qui améliore Z.

Algorithme 13 : Modification de l'étape de résolution de P_aux

Choisir un multiflot f réalisable
Poser $f' \leftarrow f$
répéter
  $f \leftarrow f'$
  calculer $Z = \max_i Z_i(f)$ et prendre $i = \text{argmax} Z$ $(Z = Z_i)$
  résoudre une instance de MFCM avec $Z = Z_i$ (donne f')
jusqu'à $f' == f$

Cet algorithme est fondé sur la résolution de problèmes de type MFCM, problèmes que nous allons détailler au chapitre suivant.

4.5. Conclusion

Dans ce chapitre, nous avons tenté de résoudre un problème de type CFEMF en appliquant la technique de décomposition de Benders. Le problème maître étant très difficile à résoudre, nous avons essayé de le simplifier en ne considérant qu'une seule coupe à la fois. Nous avons montré les lacunes de cette approche et modifié en conséquence l'heuristique proposée.

Les temps de calculs obtenus sont très mauvais dans la mesure où la résolution du programme maître à une seule coupe, baptisé P_{aux}, est obtenue grâce à la programmation linéaire. Dans le chapitre suivant, nous allons nous pencher sur la résolution efficace d'un tel problème.

Chapitre 5

Stratégies de résolution pour le problème de Multiflot Fractionnaire de Coût Minimal

Sommaire

Nous avons décrit au chapitre précédent le problème CFEMF, un problème de flot entier couplé avec un multiflot fractionnaire permettant de prendre en compte la notion de temps dans les réseaux de mobilité, avec les modèles et les méthodes de résolution exactes. Nous avons donné des pistes pour une résolution heuristique.

Le problème maître de la résolution exacte par la méthode de Benders, tout comme le problème auxiliaire du schéma heuristique inspiré de celle-ci, supposent que l'on sache résoudre le problème du Multiflot Fractionnaire de Coût Minimal. Malheureusement, sa résolution est difficile et mérite qu'on lui dédie un chapitre.

Le flot agrégé $Sum(f)$ du à l'économie d'échelle, que l'on a mis en avant au chapitre précédent, se traite particulièrement bien d'un point de vue heuristique en exprimant les

différents problèmes comme des recherches de cycles ou de chemins dans un réseau. Nous allons donc essayer de résoudre le problème de manière heuristique. Celle-ci utilise une méthode inspirée de la résolution d'un problème plus simple.

5.1. Problème FCEM

On cherche à résoudre le problème de Flot de Coût Entier Minimum (FCEM).

5.1.1. Description du problème

Le problème est le suivant :

Trouver dans un réseau $G = (V, E)$ un flot $g \geq 0$ qui **minimise** $pg + \lambda \lceil (g + g_0) \rceil$ sous les contraintes

$$MIN \leq g \leq MAX \tag{36.1}$$

$$\sum_{e \in \omega^-(v)} g_e - \sum_{e \in \omega^+(v)} g_e = 0 \qquad \forall v \in V \in K \tag{36.2}$$

$$g \in \mathbb{R} \tag{36.3}$$

connaissant deux vecteurs capacité MIN et MAX, deux vecteurs coût λ et p, positifs ou nuls, tous indexés sur E.

Pl. 36 : Problème FCEM

Le problème FCEM est **NP-difficile** ([BMQ03]). Dans la démonstration, on ramène la complexité de ce problème à celle du problème de localisation (*location problem*) qui est aussi NP-difficile.

5.1.2. Heuristique de résolution

Nous allons essayer de trouver une bonne solution à ce problème en appliquant une heuristique baptisée CYGEN. Le principe de résolution est très simple : le flot réalisable courant est amélioré, tant que c'est possible, en lui ajoutant un flot-cycle de coût négatif. L'algorithme qui permet d'exhiber un flot-cycle de coût négatif est donné au point suivant, il nécessite le calcul préalable d'une arborescence dans le graphe d'écart du graphe d'origine. L'algorithme de principe est donc le suivant :

Algorithme 14 : CYGEN - Résolution de FCEM

Retourne : flot g
Initialiser g compatible avec MIN et MAX
$Stop \leftarrow faux$
tant que non ($Stop$) **faire**
  Choisir une antiarborescence Λ d'antiracine x_0 dans le réseau G^* associé à g
  Appliquer CYCLE-SEARCH(g,Λ)
  si on a obtenu un cycle généralisé améliorant (ω, q) **alors**
    $g \leftarrow g + q\Phi_\omega$
  sinon $Stop \leftarrow vrai$
fin tant que

5.1.3. Algorithme CYCLE-SEARCH

L'algorithme CYCLE-SEARCH permet de trouver un circuit de coût négatif de longueur au moins égale à trois. Le problème de détection des circuits négatifs a notamment été étudié par [CG96, YK02, CBL01].

Pour déterminer ce circuit de coût négatif, on se place dans le graphe d'écart. Dans ce graphe, trouver des cycles de longueur 2 signifie que l'on envoie bien une certaine quantité de flot, certes, mais aussi que cette quantité de flot revient aussitôt à son point de départ. Ainsi, les cycles obtenus de ce type sont complètement inintéressants.

Le graphe doit être fortement connexe. On maintient une anti-arborescence de plus court chemins couvrant tous les sommets du graphe (notée Λ dans les algorithmes qui suivent). Cette anti-arborescence représente en fait un arbre des parents dans l'exploration du graphe.

Cet algorithme exploite un théorème dont l'énoncé est le suivant : la structure des parents est soit un arbre, soit un cycle de coût négatif (voir [CG96]). En particulier, si remplacer l'arc sortant de x dans l'arborescence par $[x, y]$ donne un cycle, alors cela veut dire que x était déjà un ancêtre de y dans l'arbre des parents (on ne peut plus remonter à la racine ni en partant de x, ni en partant de y).

Il faudra apporter un soin particulier à la gestion de cette structure notamment lors de l'identification d'un cycle (mise à jour des pointeurs et rendu mémoire effectif).

L'algorithme ne prend pas en compte les cycles négatifs élémentaires de longueur 2, ni d'ailleurs les cycles composés de cycles élémentaires négatifs de longueur 2.

Notations : On cherche un flot cycle de coût négatif dans le graphe d'écart E^*. Le graphe d'écart est défini de la manière suivante : $E^* = \left\{e, e^{-1}, e \in E\right\}$ où e^{-1} représente l'arc inverse de e, c'est-à-dire l'arc obtenu en inversant origine et destination.

On définit les fonctions suivantes, pour tout arc e et pour le flot g :

- $D_g(e,q) = qp_e + \lambda_e\left(\lceil (g+g_0)_e + q\rceil + \lceil (g+g_0)_e\rceil\right)$
- $D^*_g(e,q) = -qp_e - \lambda_e\left(\lceil (g+g_0)_e\rceil - \lceil (g+g_0)_e - q\rceil\right)$

liées aux coûts "généralisés suivants :

- $C_g(q,e) = \begin{cases} qp_e + \lambda_e\lceil q - t(g,e)\rceil & \text{si} \quad q \le s(g,e) \\ +\infty & \text{sinon} \end{cases}$
- $C_g(q,e^{-1}) = \begin{cases} -qp_e - \lambda_e\lfloor 1 + q - t(g,e^{-1})\rfloor & \text{si} \quad q \le s(g,e^{-1}) \\ +\infty & \text{sinon} \end{cases}$

On cherche à augmenter ou diminuer le flot d'une valeur maximale notée q sur le flot cycle trouvé Φ_ω. Cette valeur de q peut être prise dans un ensemble discret dit **ensemble significatif** et noté T_g, tel que :

$$T_g = \left\{t^-, t^+ \text{ pour } t \in [0,1] \cap \left(\{0,1\} \cup \{s(g,e^*), t(g,e^*), e^* \in E^*\}\right)\right\}$$

où

- $s(g,e) = MAX_e - g_e$ pour $e \in E^*$
- $s(g,e^{-1}) = g_e - MIN_e$ pour $e \in E^*$
- $t(g,e) = \lceil g_e + g_{0_e}\rceil - (g_e + g_{0_e})$
- $t(g,e^{-1}) = (g_e + g_{0_e}) - \lfloor g_e + g_{0_e}\rfloor$
- t^- est la valeur de t par valeur inférieure $t^- = \lim\limits_{\epsilon\to 0^+}(t-\epsilon)$
- t^+ est la valeur de t par valeur supérieure $t^+ = \lim\limits_{\epsilon\to 0^+}(t+\epsilon)$

$s(g,e)$ est appelée g-seuil de $e \in E^*$ et correspond à la valeur extrémale de q pour laquelle le flot $g + q\Phi_\omega$ reste compatible avec les capacités imposées. $t(g,e)$ est appelée g-barre de $e \in E^*$ et correspond à la valeur de q extrémale pour laquelle la partie entière (soit supérieure, soit inférieure) du flot g ne change pas de valeur.
On appelle *père* la fonction qui donne le père d'un nœud dans l'antiarborescence

Note 0 : On balaye l'ensemble dit significatif pour les différentes valeurs de q. Bien entendu, obtenir un cycle généralisé avec $q = 0$ est inintéressant. Si l'on obtient un tel cycle avec $q = 0^+$, cela peut être du à des erreurs de calcul, ou alors la convergence vers une solution potentielle sera très lente. On commence à chercher avec $q = 1^-$ pour obtenir, s'il existe, un flot chemin de plus grande diminution.

Note 1 : Dans le cas général, on dispose d'un multigraphe. On peut donc avoir deux arcs distincts de même origine et de même destination et avec un coût différent. Il faut bien faire la distinction pour prendre la distance la plus petite possible. L'antiracine doit être testée. Si on trouve un coût négatif, c'est que l'on a trouvé un cycle de coût négatif.

Note 2 : Pour détecter le cycle quand on cherche à insérer l'arc $[x,y]$, il suffit de remonter dans l'antiarborescence en partant de y. Si l'on trouve x sur le chemin menant à la racine, alors on vient de trouver un cycle. On peut donc écrire :
$\exists p \in \mathbb{N}/\text{père}^p(y) = \emptyset$ **OU** $\text{père}^p(y) = x$ (l'entier p est inférieur à la profondeur de l'arbre)

Algorithme 15 : CYCLE-SEARCH

Retourne : un cycle généralisé (q, Φ_ω), éventuellement rien
Construire T_g [Ensemble significatif associé à g]
$q \leftarrow 1^-$, $Stop \leftarrow faux$
tant que (**non** Stop) **faire**
  Calculer les coûts $C_g(q, e), e \in E^*$
  Calculer les distances $D_g(q, x')$ de x' à x_0 dans Λ au sens de C_g, $\forall x'$
  $L_{aux} \leftarrow \emptyset$
  $Stop_1 \leftarrow faux$
  tant que non $Stop_1$ **faire**
    tant que $\exists e = [x, y] \in E^* - \Lambda$ et $[y, x] \notin \Lambda$ et **non** $Stop_1$ et $D_g(q, x) > D_g(q, y) + C_g(q, [x, y])$ [Note 1] **faire**
      Remplacer dans Λ l'arc sortant de x par $[x, y]$
      si on obtient un cycle de coût négatif [Note 2] **alors**
        $Stop_1 \leftarrow Stop \leftarrow vrai$
        Construire le cycle associé dans G
      sinon Recalculer $D_g(q, x'), \forall x'$
    fin tant que
    si non $Stop_1$ et $([y, x] \in \Lambda - L_{aux}, [x, y] \in E^* - \Lambda)$ est un circuit négatif **alors**
      $L_{aux} \leftarrow L_{aux} \cup [y, x]$
      $e \leftarrow$ arc sortant de x
      Choisir e' de poids C_g minimal et tel que $\Lambda - \{e, [y, x]\} + \{[x, y], e'\}$ reste une antiarborescence (d'antiracine x_0 ou y suivant les cas) [Note 3]
      si y est la nouvelle antiracine **alors** $x_0 \leftarrow y$
      Recalculer $D_g(q, x'), \forall x'$
    sinon $Stop_1 \leftarrow vrai$
  fin tant que
  si toutes les valeurs de T_g n'ont pas été explorées
  alors prendre la valeur de q suivante
  sinon $Stop \leftarrow vrai$
fin tant que

Note 3 : Lorsque l'on cherche à casser les cycles de longueur 2 grâce à la liste auxiliaire L_{aux}, on obtient deux sous-arbres disjoints d'antiracine x_0 et y. On essaie alors d'obtenir une seule antiarborescence d'antiracine x_0 ou y avec un arc de coût minimal, si la connexion est possible (sinon la structure d'arbre est inchangée, mais ce cas ne doit pas se produire fréquemment).

5.2. Problème MFCM

Cette section est consacrée au problème du Multiflot Fractionnaire de Coût Minimal (MFCM). On donne deux vecteurs coût indexés sur E : $p = (p_e)_{e \in E}$ et $\lambda = (\lambda_e)_{e \in E}$. On cherche un multiflot f compatible avec les triplets (o^k, d^k, D^k) avec $k \in K$ où D^k représente la quantité de flot à router entre l'origine o^k et la destination d^k. Sans perte de généralité, on impose que $\sum_{k \in K} D^k = 1$: les demandes sont normalisées.

Le problème MFCM se formule donc ainsi :

Trouver le multiflot f qui **minimise** $p.Sum(f) + \lambda\lceil Sum(f)\rceil$
sous les contraintes

$$f^k \text{ route } D^k \tag{37.1}$$
$$f \text{ multiflot} \tag{37.2}$$

Pl. 37 : Problème MFCM

Au chapitre précédent, le vecteur coût est non nul seulement sur un sous-ensemble d'arcs $A \subset E$. Il suffit de poser $\lambda_e = 0, \forall e \in \overline{A}$.

5.2.1. Méthode de résolution exacte

Afin de pouvoir concevoir des heuristiques de résolution dédiées au problème MFCM, nous avons besoin de connaître, si possible, la solution optimale de certaines instances du problème. On se propose donc de reformuler le problème en termes de programme mixte pour que celui-ci puisse être résolu par un solveur de programmes linéaires.

Pour ce faire, il suffit d'introduire une variable binaire S_e pour chaque arc $e \in E$ dominant le flot agrégé $Sum(f)$ sur cet arc. Autrement dit, S_e vaut 1 si $Sum(f) > 0$ et 0 sinon.

On peut donc écrire le PL suivant en explicitant les contraintes de routage (les constantes b_v^k sont les constantes usuelles de conservation du flot en chaque sommet).

Trouver le flot entier S et le multiflot f qui **minimise** $\lambda.S + p.Sum(f)$
sous les contraintes

$$\sum_{e\in\omega^-(v)} f_e^k - \sum_{e\in\omega^+(v)} f_e^k = b_v^k \qquad \forall v \in V, \forall k \in K \tag{38.1}$$
$$Sum(f)_e \leq S_e \qquad \forall e \in E \tag{38.2}$$
$$f \in \mathbb{R}^+ \tag{38.3}$$

Pl. 38 : Problème MFCM en PL

La présence des variables entières (au nombre de $|A|$ ou de $|E|$) rend le problème difficile à résoudre.

Remarque :

Nous avons écrit le dual de ce PL pour voir si le problème ne pouvait pas être formulé comme un problème de tension dans un graphe. Malheureusement, même si l'expression du problème en est très proche, il reste des termes (variables) parasites qui sont difficiles à prendre en compte (il faut fixer ces variables pour obtenir un problème de tension).

5.2.2. Première heuristique : FACEM

On peut remarquer que le flot agrégé $Sum(f)$ a un rôle particulier dans l'expression du problème. On va essayer de résoudre celui-ci, non pas en manipulant le multiflot f mais le flot agrégé. L'avantage de cette méthode est immédiat : on peut utiliser les outils développés pour la résolution du problème FCEM. L'heuristique dévéloppée porte le nom de FACEM, Flot Agrégé de Coût Entier Minimal.

Si le passage du multiflot au flot agrégé est trivial, il n'en est pas de même pour retrouver le multiflot original lorsque l'on ne connaît que le flot agrégé. On va supposer que le flot agrégé est **redécomposable**, c'est-à-dire que l'on peut retrouver simplement le multiflot d'origine. Bien évidemment, cette propriété est loin d'être triviale à démontrer et à maintenir.

Hypothèse de non-bifurcation

On fera tout d'abord l'hypothèse de **non-bifurcation** sur le multiflot, c'est-à-dire que chaque composante du multiflot est un chemin simple. Concrètement, on remplace le multiflot $f = (f^k)$ recherché par l'ensemble $\Gamma = (\Gamma^k)$ où chaque Γ^k est un chemin allant de o^k à d^k. Une condition nécessaire mais pas suffisante pour assurer la décomposabilité du flot agrégé est que le flot agrégé respecte des coupes dites métriques.

Notons que l'on peut introduire facilement l'hypothèse de non bifurcation dans l'expression du problème MFCM. Il suffit de modifier les contraintes de définition du multiflot f : f^k n'est plus à valeur continue dans l'intervalle $[0, D^k]$ mais dans $\{0, D^k\}, \forall k \in K$, autrement dit des contraintes en $\{0, 1\}$. On augmente donc la complexité spatiale du problème de départ.

Introduire la propriété de non-séparation du flot ne simplifie pas le problème dans la mesure où le problème de flot k-séparable est NP-difficile [BKS02], même pour le cas où $k = 1$ [KLE96].

Coupe métrique

Soit $Z \subset V$, on définit $OD(Z) = \{k \in K / o_k \in Z \text{ et } d_k \notin Z\}$. La famille de coupes métriques s'expriment alors par :

$$\forall Z \subset V, \sum_{e \in \omega^+(Z)} g_e \geq \sum_{k \in OD(Z)} D_k$$

Ce qui peut encore se traduire par la phrase suivante : le flot total qui sort de tout sous-ensemble de nœuds est au moins égal à la demande totale générée par ce sous-ensemble par rapport au reste du réseau.

Le nombre de coupes métriques est bien entendu exponentiel (en le nombre de sommets du graphe). En effet, une coupe métrique est définie par un sous-ensemble de sommets du graphe. La coupe métrique contenant tous les sommets est trivialement vérifiée donc inintéressante, et ainsi le nombre total de coupes est $\sum_{i=1}^{n-1} C_n^i = 2^n - 2$.

Pour de plus amples explications sur les coupes métriques, on pourra se référer à [MBB01, BCGT98, OK71].

Adaptation de l'algorithme CYGEN

On reprend les notations introduites au point précédent pour l'algorithme CYGEN. On note Δ l'ensemble des coupes métriques que l'on maintient afin d'assurer la décomposabilité du multiflot. On utilise les coûts $\alpha_e = D^k.p_e + \lambda_e \left(\lceil g_e + D^k \rceil - \lceil g_e \rceil \right)$ à l'initialisation. L'algorithme est paramétré par $SEUIL$, un nombre maximal d'itérations pour lequel la quantité $\lambda \lceil g \rceil + pg$ n'a pas été améliorée.

Cet algorithme est un peu dense. Pour en faciliter la lecture, nous n'avons explicité ni la recherche de cycles, ni la recherche d'un chemin admissible, mais nous les décrivons en détail dans les paragraphes suivants.

Algorithme 16 : CYGEN-AGREG - Résolution de FACEM

Retourne : Un ensemble de chemins Γ
$\rightsquigarrow$ Initialiser g
Ordonner les commodités par quantités D^k décroissantes
$g \leftarrow 0$, $\Delta \leftarrow \emptyset$, $Stop \leftarrow faux$
pour tout $k \in K$ **faire**
  Déterminer un PCC Γ_k de o_k à d_k dans G pour les coûts α_e
  $g \leftarrow g + D^k \Gamma_k$
  Ajouter à Δ les coupes métriques associées à la construction de Γ_k (Dijkstra)
fait
Poser $\Gamma \leftarrow \{\Gamma_k, k \in K\}$
$\rightsquigarrow$ Amélioration de g par itérations successives
tant que non Stop **faire**
  Construire une antiarborescence Λ d'antiracine x_0 dans G^* associé à g
  Appliquer SEARCH-CYCLE-AGREG(g, Λ)
  si SCA produit un flot satisfaisant les contraintes de coupe **alors**
    Poser $h \leftarrow g + q\Phi_\omega$, $Stop_1 \leftarrow faux$
    tant que (non $Stop_1$) et $(g \neq h)$ **faire**
      Chercher $k \in K$ et un chemin γ de o_k vers d_k tel que remplacer Γ_k par γ induit une diminution stricte de $\sup_{e \in E} |(g-h)_e|$.
      si k et γ existent **alors**
        $\Gamma_k \leftarrow \gamma$
        Recalculer g
      sinon
        $Stop_1 \leftarrow vrai$
        Déduire pour chaque $k \in K$ une partition $(Z_k, V \backslash Z_k)$ de V associée à l'échec de la construction de γ
        $\Delta \leftarrow \Delta \cup \bigcup_{k \in K} \{$la coupe métrique associée à $Z_k\}$
      fin si
    fin tant que
    si $\lambda \lceil g \rceil + pg$ n'a pas diminué **alors**
      $compteur \leftarrow compteur + 1$
      si $compteur > SEUIL$ **alors** $Stop \leftarrow vrai$
    fin si
  sinon $Stop \leftarrow vrai$
fin tant que

Adaptation de CYCLE-SEARCH

On dispose d'un flot $g = Sum(f)$ qui vérifie un ensemble Δ de coupes métriques. On cherche un flot-cycle Φ_ω qui permet d'améliorer la fonction objectif et tel que le nouveau flot $g + q\Phi_\omega$ vérifie aussi Δ. Il suffit donc de vérifier les coupes une à une mais on peut aussi se contenter (surtout si on arrive à les identifier facilement) de ne vérifier que les coupes dont au moins un des arcs appartient au flot-cycle Φ_ω.

Cette vérification aura lieu au moment de l'identification du cycle de coût négatif. Dans l'exploration des arcs admissibles, suivant la méthode de parcours de ces arcs, on pourra envisager une liste d'arcs interdits que l'on réinitialisera à chaque changement (qui conserve les propriétés d'antiarborescence) de l'arbre des pères.

Il faut modifier l'algorithme CYCLE-SEARCH (15) en modifiant la boucle de la manière suivante :

Algorithme 17 : CYCLE-SEARCH-AGREG

```
[...]
tant que (non Stop) faire
    [...]
    L²_aux ← ∅
    Stop_1 ← faux
    tant que non Stop_1 faire
        tant que ∃e = [x, y] ∈ E* − Λ et e ∉ L²_aux et [y, x] ∉ Λ et non Stop_1 et
        D_g(q, x) > D_g(q, y) + C_g(q, [x, y]) faire
            Remplacer dans Λ l'arc sortant de x, e_0 par e
            si on obtient un cycle ω de coût négatif alors
                Construire le flot-cycle Φ_ω associé dans G
                si Φ_ω respecte les coupes métriques alors
                    Stop_1 ← Stop ← vrai
                sinon
                    L²_aux ← L²_aux ∪ e
                    Remplacer e par e_0 dans Λ
                fin si
            sinon Recalculer D_g(q, x'), ∀x'
        fin tant que
        [...]
    fin tant que
    [...]
fin tant que
```

Déterminer un nouveau chemin

Dans CYGEN-AGREG, nous avons donné succinctement ce que nous faisons pour trouver un nouveau chemin pour une commodité donnée. Nous allons revenir en détail sur ce point. L'algorithme CYCLE-SEARCH-AGREG nous donne un flot-cycle généralisé, c'est-à-dire un flot cycle Φ_ω et un réel q. Ainsi, si g est le flot agrégé initial, on peut calculer $h = g + q\Phi_\omega$ le flot agrégé optimal. L'idée est de trouver un nouveau multiflot, sous la forme d'un ensemble de chemins, dont le flot agrégé est idéalement h. On va donc modifier le multiflot courant et s'arranger pour violer le moins possible la relation d'ordre entre les deux flots : $g_e \leq h_e, \forall e \in E$.

On calcule $\aleph \subset K$ l'ensemble des commodités qui empruntent un arc dont le flot h est inférieur au flot g initial (c'est-à-dire un arc dont la copie "inverse" e^{-1} appartient au cycle ω dans le graphe d'écart).

Pour une commodité k donnée, associée à la demande D^k, on calculera l'ensemble des différences $\{\sup(0, g_e + D^k \geq h_e + \epsilon)\}_e$ où ϵ est une erreur donnée.

Algorithme 18 : Reconstruction de Γ pour CYGEN-AGREG

Retourne : Mise à jour de Γ et obtention d'une erreur ϵ
Calculer $\aleph$ et l'ordonner par demande décroissante
Calculer $g \leftarrow \sum_{k \in \overline{\aleph}} f^k$
$\epsilon \leftarrow 0$
pour tout commodité de $\aleph$ **faire**
  Calculer et ordonner $\mathbb{D}$
  pour tout $d \in \mathbb{D}$ **faire**
    si $\epsilon \leq d$ **alors** $\epsilon \leftarrow d$
    si il est possible d'aller de o^k à d^k moyennant ϵ **alors**
      mettre à jour Γ avec un tel chemin
      mettre à jour le flot agrégé associé g, $\mathbb{D} \leftarrow \emptyset$
    sinon ajouter à l'ensemble des coupes métriques la coupe rencontrée
  fait
fait

Bien évidemment, l'erreur relevée à la fin de l'algorithme, ϵ ,permet d'évaluer le multiflot Γ nouvellement calculé. Si cette erreur ϵ est supérieure ou égale à q, on peut douter de la qualité de celui-ci.

Répondre à la question de l'existence d'un chemin entre deux points est très facile, il suffit de parcourir le graphe en largeur et de marquer les sommets déjà rencontrés. Si on ne peut trouver un tel chemin, nous disposons, par définition, d'une coupe d'arcs entre ces deux sommets. Au final, on trouve toujours un chemin entre l'origine et la destination d'une commodité déterminée, seulement cette existence a un prix sur la dégradation de la solution (on peut retrouver la solution initiale que l'on voulait améliorer).

Coupes générées à l'initialisation de CYGEN-AGREG

L'algorithme de Dijkstra marque les différents nœuds et partitionne l'ensemble des nœuds en deux sous-ensembles : celui dont la distance avec le nœud de départ est connue et les autres. Comme notre graphe est fortement connexe, on est amené à générer au plus $n-2$ coupes (la dernière, contenant tous les sommets du graphe, est trivialement vérifiée). On pourrait peut-être se contenter de générer toutes les coupes sur le chemin Γ_k.

La première possibilité quand on génère les coupes associées à Γ_k est de ne considérer que la commodité k. Il est alors inutile de générer toute coupe contenant la destination. Lorsque deux coupes de Δ auront même ensemble de sommets, il suffira de les "fusionner" pour obtenir une coupe plus restrictive (c'est-à-dire prendre en compte les commodités associées).

La seconde possibilité est de considérer l'ensemble total des commodités. Il est nécessaire de vérifier que la coupe n'est pas triviale, i.e. que $OD \neq \emptyset$ (le graphe étant fortement connexe, le cocycle sortant est non vide dès lors que OD est aussi non vide). Nous retenons cette solution.

Illustrons ce choix sur un graphe triangulé à 10 sommets.

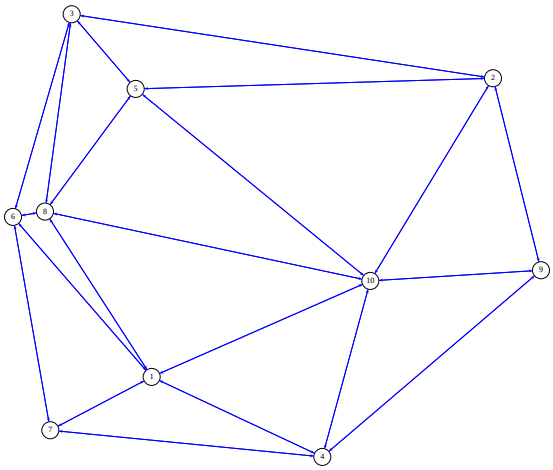


FIG. 5.1 – Exemple pour la génération de coupes métriques

L'ensemble des commodités comprend tous les couples origine-destination sur l'enveloppe convexe, ie. toutes les combinaisons des sommets deux à deux 2, 3, 4, 6, 7, 9. On s'intéresse à la génération des coupes pour la commodité d'origine 3 et de destination 2 notée (3→2) avec la distance euclidienne.

Certaines coupes sont aussi valides lorsque l'on considère d'autres commodités. Voici les sept coupes générées successivement en appliquant l'algorithme de Dijkstra. Nous donnons aussi les autres commodités qui génèrent les mêmes coupes.

- $\{3\}$, $\{3,5\}$, $\{3,5,8\}$, associées aux commodités $\{(3\to 2)(3\to 4)(3\to 6)(3\to 7)(3\to 9)\}$
- $\{3,5,6,8\}$, $\{1,3,5,6,8\}$, $\{1,3,5,6,8,10\}$, associées aux commodités $\{(3\to 2)(3\to 4)(3\to 7)(3\to 9)(6\to 2)(6\to 4)(6\to 7)(6\to 9)\}$
- $\{1,2,3,5,6,8,10\}$ associée aux commodités $\{(2\to 4)(2\to 7)(2\to 9)(3\to 4)(3\to 7)(3\to 9)(6\to 4)(6\to 7)(6\to 9)\}$

5.2.3. Seconde heuristique

Le principe de cette heuristique est très simple : on choisit un arc e du graphe qui porte du flot ($Sum(f_e) > 0$). On diminue le flot sur cet arc en appliquant un algorithme de flot de coût minimum, composante du multiflot par composante, ie en traitant une commodité à la fois. On réitère le processus tant que nécessaire.

La fonction objectif à minimiser est de la forme : $p.Sum(f) + \lambda\lceil Sum(f)\rceil$. Pour chaque arc e portant du flot, il faut prendre en compte un coût p_e linéairement dépendant du flot total porté par l'arc et il faut aussi prendre en compte le coût d'utilisation λ_e de l'arc e. Ce coût ne dépend pas linéairement du flot que l'arc porte mais il intervient dès lors que l'arc est utilisé. Ce coût est difficile à modéliser : en effet, lorsque l'on travaille sur une composante du multiflot, il n'y a pas de surcoût supplémentaire si l'arc est déja utilisé ; en revanche, si l'on souhaite annuler ce coût, il est nécessaire d'annuler le flot de toutes les composantes qui utilisent cet arc.

L'idée est donc de approximer la fonction coût originale, une marche d'escalier par une fonction coût continue, comme le montre la figure 5.2. Nous avons choisi la fonction $\lambda\frac{\alpha x}{\alpha x+1}$, la paramètre α est un paramètre permettant de contrôler la pente à l'origine, on le choisit tel que la fonction coût soit une fraction donnée r de λ en une abscisse donnée. Pour avoir un coût en λ de même nature que le coût en p, on utilise la dérivée $\lambda\frac{\alpha x}{(\alpha x+1)^2}$.

Remarquons que suivant la valeur de λ, les coûts calculés peuvent être très importants et donc que cela peut générer une instabilité numérique, d'où l'intérêt d'utiliser le paramètre α.

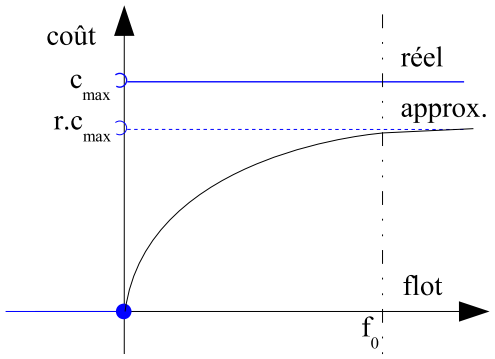


FIG. 5.2 – Coût d'un arc et approximations

Nous utilisons une mesure d'utilité des arcs, c'est-à-dire le nombre de composantes du multiflot qui sont non nulles sur un arc donné. On note cet indice u_e pour un arc $e \in E$ donné : $u_e = \sum_k \lceil f^k e\rceil, e \in E$.

La première étape de l'algorithme est de choisir un arc e de E sur lequel appliquer la réduction de flot. Différents critères ont été utilisés :

- l'arc est choisi dans une liste cyclique, liste qui contient les arcs où le multiflot est non nul (1),
- $\max_{e\in E|u_e>0}\{\lambda_e\}$, l'arc qui a le plus grand coût d'utilisation,
- $\max_{e\in E|u_e>0}\{\frac{\lambda_e}{u_e}\}$, l'arc qui a le plus grand coût d'utilisation ramené au nombre de compo-

santes qui l'utilise (le plus mauvais rapport "qualité prix") (2),

- $\max_{e \in E | u_e > 0} \{\lambda_e . u_e\}$,
- $\max_{e \in E | u_e > 0} \{\lambda_e + p_e\}$,
- $\max_{e \in E | u_e > 0} \{p_e + \frac{\lambda_e}{u_e}\}$,
- $\max_{e \in E | u_e > 0} \{p_e + \lambda_e . u_e\}$,
- l'arc est choisi au hasard parmi tous les arcs porteurs de flot du graphe.

Les expérimentations nous ont montré que les méthodes les plus prometteuses sont (1) et (2).

Une fois l'arc choisi, nous appliquons un algorithme de flot de coût minimal sur chaque composante du multiflot. Cette recherche de flot est paramétrée par la contrainte de capacité sur l'arc choisi, nous avons essayé différents cas de figure :

- aucune contrainte de capacité,
- capacité maximale nulle (flot nul sur e),
- une capacité maximale égale à une fraction du flot préalablement porté.

La méthode la plus efficace se révèle être d'imposer la nullité du flot sur l'arc. L'ordre dans lequel ont été prises les composantes n'a jamais été changé.

La condition d'arrêt de l'algorithme est un certain nombre d'itérations ($iter_{\max}$) à l'issue desquelles la fonction d'évaluation du flot n'a pas été améliorée. Nous avons par exemple pris $|E| = m$ itérations non améliorantes avec la politique de choix d'arc par liste cyclique.

L'algorithme de coût minimum par composantes est donc le suivant : il retourne la valeur de la fonction objectif $Z_{\min}$ et le multiflot $f_{\min}$ correspondant.

Algorithme 19 : Algorithme de coût minimum par composantes

Entrée : multiflot f compatible avec le routage
Retourne : un multiflot $f_{\min}$ associé au coût Zmin
Initialiser le multiflot f, $f_{\min} \leftarrow f$
$Stop \leftarrow$ **faux**, $iter \leftarrow 0$
tant que (**non** Stop) **faire**
  choisir un arc e
  appliquer l'algorithme de coût minimal à e sur chaque composante de f suivant les coûts définis à la figure 5.2
  mettre à jour f
  calculer la valeur objective Z associée
  si $Z < Z_{\min}$ **alors**
    $Z_{\min} \leftarrow Z$, $f_{\min} \leftarrow f$
    $iter \leftarrow 0$
  fin si
  $iter \leftarrow iter + 1$
  si $iter = iter_{\max}$ **alors** $Stop \leftarrow$ **vrai**
fin tant que

L'initialisation du multiflot f se fait en résolvant le PL où les contraintes d'intégrité de S sont relâchées.

Évolution de la fonction-objectif en fonction du paramètre (α)

La valeur de la fonction-objectif peut varier fortement en fonction de la valeur choisie du paramètre α comme le montre les exemples suivants (résolution de deux instances avec choix de l'arc dans une liste).

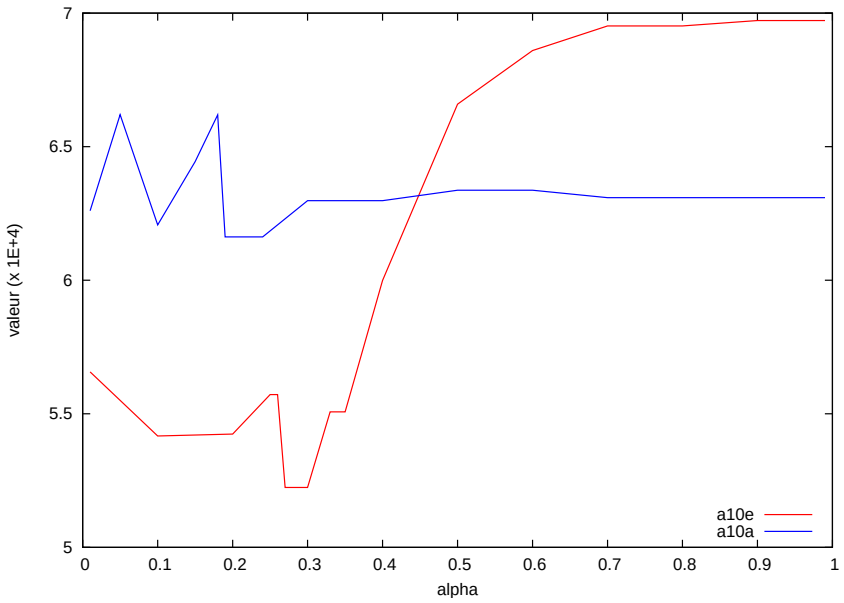


FIG. 5.3 – Variation de la valeur finale en fonction de α

Dans le processus de résolution, il est donc nécessaire de tester différentes valeurs pour α. On peut se contenter d'un ensemble fini de valeur sur l'intervalle $[0, 1[$ ou bien mettre en place une analyse un plus plus fine sur certains segments prometteurs.

5.2.4. Comportement de MFCM dans CEFMF

On tente de résoudre le problème CEFMF par la méthode de Benders où le problème maître est une instance du problème MFCM (sections 4.4.3 et 4.4.3). Nous rappelons que l'on dispose de la solution optimale pour certaines instances grâce à la résolution du problème par un solveur de programme linéaire comme CPLEX. Connaître cette solution permet d'évaluer nos heuristiques et d'étudier leur comportement.

Le comportement de l'heuristique n'est pas très bon. Dans le cas général, on trouve bien une condition d'arrêt (le multiflot f est inchangé par MFCM) mais malheureusement la valeur obtenue par cette méthode est très loin de l'optimum.

D'après A. Geoffrion, on peut se contenter d'une solution sous-optimale pour le PM. Or, la solution optimale donne une borne inférieure sur la solution finale du problème. Bien évidemment, si le PM n'est pas résolu à l'optimalité, on ne garantit plus l'obtention d'une telle borne. Il est fort probable que la solution obtenue actuellement ne soit pas satisfaisante, c'est-à-dire que la solution est trop mauvaise. Nous sommes convaincus que pour qu'une solution soit satisfaisante, cette solution doit être tout de même une borne inférieure de la solution s'il s'agit d'une minimisation (même si la valeur de moins bonne qualité que celle que l'on peut obtenir avec la résolution exacte).

On donne un aperçu de ce que l'on obtient. On repère la valeur de la fonction-objectif optimale Z_{opt}, les Z_i représentent les différentes bornes inférieures que l'on obtient avec une résolution satisfaisante, les Z'_i sont les valeurs obtenues avec notre schéma de résolution.

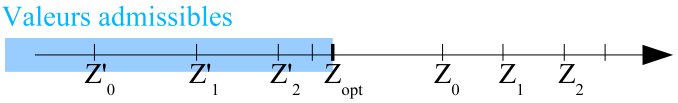


FIG. 5.4 – Itérations de Benders

Lors du processus de résolution de Benders, les différents PM, résolus à l'optimalité, donnent une suite croissante de bornes inférieures de la solution (on peut avoir à éliminer les valeurs trop petites par rapport à celles déjà obtenues). Dans la mesure où la première contrainte donne déjà une très mauvaise valeur originale pour cette suite, nous avons tenté de l'enlever dès la deuxième itération. Cela nous a permis de nous rapprocher de la solution optimale mais ce n'est pas encore suffisant pour exhiber la solution optimale.

5.2.5. Expérimentations numériques

Nous allons illustrer les différentes méthodes que nous avons utilisées pour tenter de résoudre le problème CFEMF. Cette comparaison va se faire sur les deux types de graphe que l'on a utilisé tout au long de cette thèse : le graphe à 10 sommets et le graphe à 20 sommets.

Les différentes instances, outre le support du graphe, sont paramétrées par les coûts sur les arcs. La méthode CYGEN-AGREG peut avoir différents comportements suivant le type de remontée des informations des coupes de Benders que l'on a défini au chapitre précédent. Il y a trois types : la moyenne (*moy*), le maximum (*max*), ou l'amnésie (la dernière coupe uniquement ou *last*)

Nous donnons aussi la distance à l'optimum de la meilleure valeur trouvée grâce à CYGEN-AGREG (repérée en gras).

instance	ratio	solution exacte	CYGEN-AGREG *moy*		CYGEN-AGREG *max*	CYGEN-AGREG *last*	
a10e	0.02	41275	**47474**	[15%]	61832	53600	
a10e	0.33	77614	78048		**77614**	bascule	
a20e	0.02	44051	68563		76282	**64999**	[47%]
a20e	0.33	76282	78094		**76282**	bascule	

TAB. 5.1 – Comparaison entre méthode exacte et CYGEN-AGREG

Sur l'instance *a10e*, toutes les résolutions, exactes (CPLEX, Benders) comme approchées (CYGEN-AGREG) sont instantanées.

Sur l'instance *a20e*, pour le ratio $r = 0.33$, il faut une minute pour résoudre le problème de manière exacte, la résolution avec CYGEN-AGREG prend moins de 2 secondes et est à l'optimum. Pour le ratio $r = 0.02$, résoudre le problème de manière exacte prend quasiment 6 heures alors que CYGEN-AGREG renvoie un résultat après 3 secondes.

Comme on l'a déjà remarqué au chapitre précédent, nous n'obtenons pas de solution satisfaisante (effet bascule ou trop grande distance à l'optimum) dans certains cas. La deuxième

heuristique que nous avons proposée est alors, dans ce cas bien précis, meilleure que CYGEN-AGREG mais reste très éloignée de la solution optimale déterminée par un solveur de programmes entiers mixtes, raison pour laquelle elle ne figure pas dans ce tableau de comparaison.

5.2.6. Conclusion

Nous avons présenté dans ces deux derniers chapitres, le cheminement pour disposer d'une heuristique capable de résoudre un problème NP-difficile, le problème CFEMF, un problème de couplage de flot entier et de flot fractionnaire.

Au chapitre précédent, nous avons proposé une heuristique fondée sur la méthode de décomposition de Benders pour laquelle il est nécessaire de disposer d'une méthode rapide pour résoudre le problème maître restreint à une contrainte, le problème MFCM. La résolution optimale n'est pas demandée, il suffit de trouver une bonne solution.

L'objectif de ce dernier chapitre était donc d'obtenir des solutions rapidement pour ce problème, même si elles ne sont pas optimales. Nous pouvons donc dire que notre démarche a partiellement réussi dans la mesure où nous obtenons des solutions très rapidement sur le problème CFEMF grâce à l'heuristique CYGEN-AGREG pour le problème MFCM mais ces solutions ne sont pas toujours de bonne qualité.

La résolution de MFCM reste donc un problème ouvert. Il est nécessaire d'affiner la résolution de MFCM pour améliorer l'heuristique globale de résolution de CEFMF qui gagnera ainsi en robustesse.

Conclusion

L'objet de cette thèse était de proposer des méthodes et des outils pour la conception stratégique de réseaux de transports publics en milieu urbain. Cette recherche prospective nous a permis d'inscrire cette problématique dans un cadre plus large, celui des problèmes de synthèse de réseaux de mobilité.

Cette classe de problèmes — dont certains sont à l'origine de la recherche opérationnelle — bénéficie actuellement du dynamisme lié au développement des Télécommunications. Nous avons défini les réseaux de mobilité comme des réseaux sur lesquels circulent différents produits. Ces produits doivent être acheminés de leur origine à leur destination avec un coût contrôlé (des contraintes budgétaires par exemple). Pour cela, il s'agit de concevoir un système de transport, c'est-à-dire de décrire un ensemble de tournées de conteneurs qui offrent aux produits un acheminement mutualisé plus rapide que le déplacement standard.

Nous avons tout d'abord fait un rappel des problèmes de synthèse de réseaux et un panorama des différentes méthodes et outils classiquement utilisés pour leur résolution. Nous avons aussi fait un état de l'art d'un problème de synthèse de réseaux particulier : les problèmes de tournées de véhicules.

Cette recherche bibliographique nous a permis, dans un deuxième temps, de formaliser la notion de problèmes de synthèse de réseaux de mobilité avec demande élastique. Cette notion traduit le fait que la demande d'accès au réseau de transport est sensible à la qualité de service offerte par le système (elle est d'autant plus grande que la qualité de service est bonne). Nous avons proposé un modèle général de ce type de problème où l'on a évacué la problématique temps, hypothèse simplificatrice qui peut être légitime sur les réseaux à forte densité comme les réseaux urbains. Ce modèle a été décliné en deux modèles plus simples, linéaires, afin d'étudier leur comportement sur des petites instances résolues grâce à un solveur de programmes entiers mixtes.

Nous avons proposé différents compléments ou extensions à ces modèles, une véritable boite à outils de modèles dans laquelle puiser, suivant les objectifs recherchés ou les contraintes à respecter : identification d'une ligne unique, prise en compte de temps d'attente ou de service, la desserte d'un point particulier d'un point du réseau, un mode de comportement alternatif dans le déplacement des produits.

La complexité de ces problèmes ainsi que la difficulté de résolution liée à la taille des instances réalistes nous a amené à mettre en œuvre une résolution à l'aide de métaheuristiques classiques : une méthode de type GRASP, une autre de type Tabou que nous avons finalement

hybridées pour bénéficier des avantages de chacune. L'adaptation de ces heuristiques de recherche locale font usage d'opérateurs de voisinage adaptés à une description particulière des tournées que l'on a baptisée géodésique. Une géodésique est définie par la donnée de points particuliers du circuit, les points de contrôle, et une relation entre ces points, un plus court chemin par exemple. Bien évidemment, le nombre de points de contrôle conditionne la taille et la forme, relativement simple, des tournées ainsi décrites.

Nous avons exploité les résultats du chapitre précédent pour valider notre approche sur des instances de taille relativement modeste. Nous pouvons retrouver les résultats issus de la résolution exacte avec nos heuristiques dans certaines conditions, notamment lorsque la contrainte de coût sur le système est primordiale, et être très proches des résultats optimaux dans les autres.

Nous nous sommes aussi intéressés à l'accélération d'une opération très coûteuse dans le processus de résolution, l'évaluation de la fonction objectif, en introduisant la notion de fonction auxiliaire que l'on substitue à la fonction objectif originale durant certains processus de la résolution. Cette nouvelle fonction est définie grâce à des critères topologiques et se révèle plus rapide à évaluer en faisant une abstraction, toute relative, des commodités définies sur le réseau. Cette approche permet d'obtenir de bonnes solutions encore plus rapidement au prix d'une légère dégradation de leur qualité. En tout état de cause, l'obtention de solutions de bonne qualité est déjà satisfaisante dans la mesure où, pour des instances de grande taille, on ne peut obtenir de résultats avec un solveur de programmes linéaires mixtes et ceci même sur les problèmes où l'on a relâché les contraintes d'intégrité.

Dans les deux derniers chapitres, nous avons cherché à réintroduire la problématique temps que l'on avait éludé dans les approches précédentes. Cette problématique est complexe et recouvre, par exemple, des contraintes comme la date d'arrivée au plus tard pour les différentes commodités définies sur le réseau. Pour cela, nous avons présenté une modélisation d'un réseau temporisé discret qui nécessite de résoudre un problème très général de couplage entre un flot entier et un multiflot fractionnaire. Nous avons mis en œuvre une technique de résolution fondamentalement différente des métaheuristiques utilisées jusqu'à présent. Nous avons proposé une heuristique inspirée de la méthode de décomposition de Benders centrée sur la synthèse de l'information du routage des produits. Le problème maître restreint "classique" étant trop difficile à résoudre, nous nous sommes intéressés au problème à une seule contrainte, le problème MFCM. L'heuristique offre des résultats proches de l'optimum mais il est nécessaire de résoudre rapidement MFCM. L'algorithme CYGEN-AGREG tente de résoudre ce problème NP-difficile mais il n'apporte pas tous les résultats escomptés. Il nécessite donc d'être approfondi pour améliorer la précision et la robustesse du processus global de résolution.

Les perspectives ouvertes par ces travaux de recherche sont bien entendu idéalement liées à une mise en œuvre de ces outils de modélisation et de résolution sur des instances pratiques issues, par exemple, du monde des Transports et étudier leur impact sur des réseaux déjà existants et les bénéfices qu'une société exploitante pourrait en tirer.

Un retour d'expérience par des professionnels permettrait de juger de la pertinence de certaines hypothèses peut-être trop simplificatrices. Cela nous permettrait aussi d'approfondir

les travaux préliminaires que nous avons menés concernant les couplages entre les méthodes d'optimisation et l'évaluation par simulation des solutions trouvées et notamment l'enrichissement de modèle que nous avons présenté en annexe. Ce type de couplage permet en effet d'allier la puissance d'une méthode d'optimisation sur un modèle simple avec le "réalisme" de l'évaluation plus complexe par la simulation. Les résultats sont prometteurs et semblent être plus en adéquation avec les modèles réalistes.

Dans le dernier chapitre d'annexe, nous avons proposé une première réflexion sur certaines pratiques actuelles dans le développement d'applications de recherche opérationnelle et à en proposer des schémas de conception. Ces schémas utilisent à la fois la programmation orientée objet et la programmation générique pour obtenir des composants réutilisables et efficaces. Nous avons étudié les raisons de cette inefficacité et donné des solutions idoines.

Il semble maintenant important de poursuivre dans la formalisation d'une méthodologie de développement pour ce type d'application. Une des voies qui semble la plus prometteuse est le concept, récent, d'instanciation partielle des composants génériques dont le mécanisme induit est déroulé au moment de la compilation et non pas lors de l'exécution comme pour le mécanisme d'héritage.

Nous avons posé, dans cette thèse, les fondements pour réaliser un véritable environnement d'aide à la décision : tant au niveau de la résolution des problèmes que de la conception et l'implémentation informatique. Le simulateur dispose même d'une véritable interface graphique qui permet de visualiser le déplacement des objets, conteneurs et produits, sur le réseau. Il serait très intéressant de mener à bien un tel projet en facilitant l'intervention de l'organe décisionnel dans toutes les étapes du processus de configuration (ou de reconfiguration) du système de transports, en proposant des post-optimisations et diverses solutions suivant certains critères comme la robustesse. Il serait aussi nécessaire d'affiner les modèles — nous avons donné par exemple des modélisations qui peuvent faire intervenir des fonctions multiobjectif — et de chercher, par conséquent, des méthodes de résolution encore plus performantes pour qu'elles conviennent aux professionnels.

Annexe A

Graphes & Algorithmes

Sommaire

Ce chapitre emprunte le nom du célèbre ouvrage de Gondran et Minoux. Le but de ce chapitre est de faire, en quelques pages, de brefs rappels sur les notions de base utilisées dans cette thèse concernant les graphes et les réseaux. Nous donnons aussi quelques algorithmes, classiques, pour les différents problèmes de plus courts chemins. Il n'y a aucune intention d'exhaustivité. Pour de plus amples renseignements, on pourra consulter [GM95] bien évidemment, [AMO93], ou encore [BT97].

A.1. Graphe

On définit un **graphe** $G = (V, E)$ par la donnée :

- d'un ensemble $V = \{v_1, \ldots, v_n\}$ dont les éléments v_i sont appelés **sommets** ou **nœuds**. On notera $|V| = n$ le nombre des sommets du graphe. Par abus de langage, on désignera un sommet par son indice ($v_i = i$).
- d'un ensemble E dont les éléments $e \in E$ sont des couples ordonnés de sommets appelés **arcs**. On note $|E| = m$. Si $e = (i, j)$ est un arc de G, on appelle i l'**origine** de l'arc et j la **destination**.

Si le graphe est non orienté, c'est-à-dire si l'on ne fait pas la distinction entre l'arc (i, j) et l'arc (j, i), dit que (i, j) est une **arrête**.

On dit que deux arcs (ou deux arrêtes) sont **adjacents** s'ils ont au moins une extrémité en commun.

On appelle **cocycle** $\omega(A)$ d'un ensemble de sommets A les arcs qui ont une extrémité dans A et l'autre dans $\overline{A}$. On appelle cocycle d'un sommet v l'ensemble des arcs incidents à ce sommet. On note ω^+ le cocycle sortant (i.e. l'ensemble des arcs sortants) et ω^- le cocycle entrant (ie.l'ensemble des arcs entrants).

Un graphe est **connexe** si tous les sommets du graphe sont reliés par au moins un chemin. Il est dit fortement connexe s'il existe un chemin orienté entre toute paire de nœuds de ce graphe.

On appelle **arbre** tout graphe connexe sans cycle. On parle d'**arborescence** pour un arbre orienté avec une seule et unique **racine**. On obtient une **anti-arborescence** lorsque l'on inverse tous les arcs d'une arborescence. La racine s'appelle alors une **anti-racine**.

Une **coupe** est une partition de l'ensemble des nœuds d'un graphe en deux sous-ensembles § et $\overline{S}$. Chaque coupe définit un ensemble d'arcs qui ont une extrémité dans S et l'autre dans $\overline{S}$. Par extension, on appelle coupe cet ensemble d'arcs.

A.2. Flot

Soit un graphe $G = (V, E)$ connexe. On pose $|E| = m$.

On dit que le vecteur $f = (f_1, \ldots, f_m) \in \mathbb{R}^m$ est un flot (ou encore *flot simple* entre une origine $o \in V$ et une destination $d \in V$ si et seulement si la première loi de **Kirchhoff**, encore appelée **loi de conservation de flot** aux nœuds, est vérifiée pour tous les nœuds du graphe autres que l'origine et la destination.

La composante f_j de l'arc j est appelée **quantité de flot** ou **flux** sur l'arc j.

La loi de conservation s'énonce ainsi : la somme des flux entrant en un sommet est égale à la somme des flux sortants. Elle s'écrit :

$$\sum_{e \in \omega^-(i)} f_e = \sum_{e \in \omega^+(i)} f_e$$

où $\omega^-(i)$ (resp. $\omega^+(i)$) est le cocycle entrant (resp. sortant) du sommet i.

Si le flot véhicule la quantité D, on écrit $\sum_{e \in \omega^-(t)} f_e = \sum_{e \in \omega^+(s)} f_e = D$

Un flot peut avoir à respecter des contraintes de capacité (minimale et/ou maximale). Un tel flot est alors dit **compatible** ou encore **réalisable**.

Parfois, on considère un flot sans origine ni destination. On écrit juste la conservation de flot en chacun de nœuds.

A.3. Multiflot

Parfois, la notion de flot *simple* est insuffisante. Elle n'est pas adaptée au routage d'un grand nombre de produits surtout si ceux-ci ont des origines ou des destinations multiples car on ne peut synthétiser l'information dans un seul vecteur !

Si K produits circulent sur le réseau, on appelle **multiflot** l'objet correspondant à $f = (f^k)^{k\in K} = (f_e^k)_{e\in E}^{k\in K}$ où chacune des composantes f^k est un flot.

A.4. Réseau de transport

On appelle **réseau** tout graphe auquel on a associé des valeurs aux arcs comme, par exemple, une longueur ou une capacité, ...

On appelle **réseau de transport** un graphe $G = (V, E)$ où chaque arc $e \in E$ est muni d'une capacité $c_e \geq 0$. Lorsqu'un flot f circule sur G, la contrainte de capacité est vérifiée :

$$f_e \leq c_e, \quad \forall e \in E$$

On considère deux sommets particuliers de V : un sommet **source** s et un **sommet** puits t. L'arc $(t, s$ est dit arc de retour.

Le vecteur $f = (f_1, \ldots, f_m)$ est un **flot de s à t** dans G si et seulement si les lois de conservation aux nœuds sont vérifiées en tous les sommets de G sauf en s et t où l'on a :

$$\sum_{e\in\omega^-(t)} f_e = \sum_{e\in\omega^+(s)} f_e = z$$

La loi de conservation du flot porte aussi le nom de loi de KirschhofF (par analogie avec les lois électriques)

La quantité z est appelée **valeur du flot**.

Le flot est représentatif d'une certaine quantité d'une chose que l'on veut router sur ce réseau, que l'on appelle **commodité**, par exemple : des marchandises, des produits, des personnes ou des véhicules.

A.5. Multiflot

Considérons un graphe $G = (V, E)$ orienté sur lequel circulent indépendamment K flots simples $f^1, \ldots, f^K$ positifs. A k fixé, $f^k = (f_e^k)$ est un flot "simple" entre une source s_k et un puits t_k de valeur z^k.

Considérons maintenant le vecteur $F = (F_e)_{e\in E}$ défini par $F_u = \sum_{k=1}^{K} f_u^k$. On dit que F est un **multiflot** sur G de valeurs $z_1, \ldots, z^k$. La valeur totale de F est $Z = \sum z^k$.

A.6. Lemme de Minty

Soit un graphe muni d'une 4 coloration sur les arcs (vert, noir, bleu et rouge par exemple). Minty (voir [MIN61]) a montré qu'une et une seule des propositions suivantes est vérifiée pour tout arc $u = (x, y)$ du graphe.

- L'arc u appartient à un cycle constitué uniquement (à l'exception de u lui-même) d'arcs verts, noirs et bleus avec les arcs noirs orientés dans le même sens que u, les arcs bleus dans le sens opposé. L'orientation des arcs verts n'est pas importante.
- L'arc u appartient à un cocycle constitué uniquement (à l'exception de u) d'arcs rouges, noirs et bleus avec les arcs noirs orientés dans le même sens que u, les arcs bleus dans le sens opposé. L'orientation des arcs rouges est indifférente.

A.7. Algorithmes de Plus Courts Chemins classiques

Dans cette section, nous allons exposer quelques algorithmes de calcul de plus courts chemins suivant leur cadre d'utilisation (hypothèses, finalité). Il n'y a aucune volonté d'exhaustivité ou de classification.

On pourra se reporte au rapport technique de Pallottino et Scutella [PS97] pour un état de l'art des Plus Courts Chemins pour les transports.

Nous allons calculer les plus courts chemins au sens d'une certaine fonction coût c. Chaque arc $e \in E$ sera muni d'un poids c_e. La fonction c sera étendu de E sur V^2 comme suit :

$$c_{ij} = \begin{cases} c_{ij} \text{ si } (i,j) \in E \\ 0 \text{ si } i = j \\ +\infty \text{ sinon} \end{cases}$$

A.7.1. Algorithme de Moore-Dijkstra

C'est un des algorithmes de plus courts chemins les plus faciles à comprendre, à implémenter et aussi l'un des plus rapides.

But : Calculer les plus courts chemins d'un sommet à tous les autres

Résultat :

- les valeurs des plus courts chemins du sommet source à tous les autres sommets (accessibles depuis ce sommet),
- l'arbre des plus courts chemins dont la racine est le sommet source

Hypothèses : Tous les coûts doivent être positifs

Complexité : Elle est en $O(n.\log n)$ avec une structure de données adaptée (un *tas min* pour ne pas la nommer).

Algorithme :

L'algorithme maintient deux ensembles de sommets, complémentaires l'un de l'autre : d'un côté, les sommets dont la distance à la source est connue et minimale et de l'autre les sommets qu'il reste à explorer.

Algorithme 20 : Moore-Dijkstra

```
⇝ initialisations
S ← ∅, S' ← V
∀x ∈ V,   π_x ← +∞,   pred_x ← ∅
π_s ← 0, sortie ← faux
⇝ marquage des sommets
tant que (S' ≠ ∅) et (non sortie) faire
    choisir x ∈ S' tel que π_x soit minimal (π_x ≤ π_y,   ∀y ∈ S')
    sortie ← (π_x = +∞)
    S ← S ∪ {x}
    S' ← S' − {x}
    pour tout (x, y),   y ∈ S' faire
        π'_y ← π_x + c(x, y)
        si (π'_y < π_y) alors
            pred_y ← x
            π_y ← π'_y
        fin si
    fait
fin tant que
Déterminer le PCC en remontant les prédécesseurs en partant de d
```

L'algorithme ne donne pas explicitement les plus courts chemins, juste leur valeur. Il faut remonter les prédécesseurs.

A.7.2. Algorithme de Floyd-Warshall

But : Calculer les plus courts chemins entre tous les sommets [GSF87].

Résultat :

- D est la matrice des plus court chemins
- P est la matrice des prédécesseurs encore appelée matrice des liaisons.

Hypothèses : Le graphe ne doit contenir aucun circuit absorbant (i.e. de coût négatif)

Complexité : Cet algorithme est en $O(n^3)$. Dans le cas où le graphe est "creux", i.e. $m \ll n^2$, il peut être intéressant d'utiliser d'autres algorithmes comme l'algorithme précédent — celui de Dijkstra — qui est en $O(m.n \log n)$.

Algorithme :

Algorithme 21 : Floyd-Warshall

```
⇝ initialisations
pour i de 1 à n faire
    pour j de 1 à n faire
        d_ij ← c_ij
        p_ij ← i
    fait
fait
pour k de 1 à n faire
    pour i de 1 à n faire
        pour j de 1 à n faire
            si (d_ik + d_kj ≤ d_ij) alors
                d_ij ← d_ik + d_kj
                p_ij ← p_kj
            fin si
        fait
    fait
fait
retourner (D,P)
```

Ce même algorithme peut servir aussi pour déterminer si un réseau contient au moins un circuit absorbant. Il suffit de modifier le test effectué dans la boucle de mise à jour des potentiels. Si le graphe contient un circuit absorbant, les potentiels ne sont jamais optimaux, et l'algorithme s'arrête lorsque au moins un des critères ci-dessous est vérifié :

- un potentiel devient inférieur à un seuil donné,
- un potentiel n'est toujours pas optimal alors que l'algorithme a déjà dépassé les n^3 itérations (où n est le nombre de sommets du graphe).

Le sommet dont le potentiel a un comportement "anormal" est un sommet du circuit de coût négatitf.

A.7.3. Algorithme des k Plus Courts Chemins

But : Calculer les k PCC entre un sommet et tous les autres

Résultat : On obtient la valeur des k plus courts chemins entre le sommet source et tous les autres sommets accessibles depuis le sommet source. On dispose aussi des informations nécessaires au calcul explicite des chemins correspondants.

Hypothèses : Les hypothèses de l'algorithme de k PCC sont les mêmes que celle de l'algorithme qui sert de base. L'algorithme proposé ici est inspiré de l'algorithme de Dijkstra, il faut donc que tous les coûts soient positifs.

Algorithme : L'algorithme travaille sur un ensemble de nœuds "élargi" V'. Cet ensemble correspond à un nœud et à son potentiel. C'est ce nœud étendu auquel on attache le potentiel (noté π) et le prédécesseur (noté ξ) dans l'arbre de parcours des PCC (unique avec la définition du nœud étendu). Le nœud original est lié aux nœuds élargis (un pour chaque potentiel possible) par la fonction h et à son inverse h^{-1}. Pour chaque nœud du réseau d'origine, il suffit de compter le nombre (C) de chemins déjà établis.

Algorithme 22 : Généralisation de Dijkstra pour le k-PPC

```
⇝ initialisations
∀v ∈ V,   C_v ← 0
elm ← 1,h(elm) ← s, h^{-1}(s) ← {elm}, π_elm ← 0
X ← {elm}, P ← ∅
⇝ marquage des sommets
tant que (C_d < K) et (X ≠ ∅) faire
    choisir x ∈ X tel que π_x soit minimal (π_x ≤ π_y,   ∀y ∈ X)
    X ← X − {x}
    i ← h(x)
    C_i ← C_i + 1
    si (i = d) alors
        p ← chemin de 1 à x
        P ← P ∪ {h(p)}
    fin si
    si (C_i ≤ K) alors
        pour tout (i, j),   j ∈ E faire
            elm ← elm + 1
            π_elm ← π_k + c_ij, ξ_elm ← x
            h(elm) ← j, h^{-1}(j) ← h^{-1}(j) ∪ {elm}
            X ← X ∪ {elm}
        fait
    fin si
fin tant que
retourner P
```

Pour l'implémentation, les fonctions h et h^{-1} ne sont pas nécessaires. On ajoute aussi au nœud étendu, l'arc prédécesseur... En fait, en guise de fonction h, on mémorise le nœud réel

associé au nœud étendu.
Le problème des k plus courts chemins est un problème très actuel [MPS98, EPP, JM99, GCM+01, HMS03].

A.7.4. Tous les Plus Courts Chemins entre deux sommets

L'algorithme présenté ici permet de calculer tous les plus courts chemins (de même valeur) entre deux nœuds donnés (s'il existe au moins un chemin les reliant). C'est une version généralisée de l'algorithme de Moore-Dijkstra qui s'inspire aussi des travaux de Martins [MPS98].

Différences avec l'algorithme classique : Dans cette version, on ne se contente pas de gérer un seul prédécesseur par sommet mais au contraire tous les prédécesseurs possibles.

Algorithme de principe

Les données de l'algorithme sont : le graphe $G = (V, E)$, un nœud source s, un nœud destination d et une fonction coût c sur les arcs. Pour tout $x \in V$, on note π_x le potentiel de ce sommet et $pred_x$ l'ensemble des prédécesseurs, i.e. la liste des sommets qui permettent d'accéder à ce sommet avec le potentiel π.

Algorithme 23 : Tous les Plus Courts Chemins

```
⇝ initialisations ∀x ∈ V,   π_x ← +∞,   pred_x ← ∅
S = {s}, π_s ← 0
sortie ← faux

⇝ marquage des sommets
tant que (S ≠ ∅) et (non sortie) faire
    choisir x ∈ S tel que π_x soit minimal (π_x ≤ π_y, ∀y ∈ S)
    S ← S − {x}
    si (π_x > π_d) alors sortie ← vrai
    sinon si (x ≠ d) alors
        pour tout (x, y) ∈ E faire
            π'_y ← π_x + c(x, y)
            si (π'_y < π_y) alors pred_y ← ∅
            si (π'_y ≤ π_y) alors
                pred_y ← pred_y ∪ {x}
                S ← S ∪ {y}
                π_y ← π'_y
            fin si
        fait
    fin si
fin tant que
Déterminer les PCC en remontant les prédécesseurs en partant de d
```

Remarque 1 : La phase de détermination des PCC n'est pas complètement triviale. Il faudra en calculer la complexité afin de déterminer si elle ne dégrade pas la complexité donnée par un algorithme type Dijkstra.

Remarque 2 : On peut noter que l'on calcule tous les potentiels du noœud source à tous ceux qui sont accessibles. On peut donc modifier très facilement cet algorithme pour obtenir tous les PCC du nœud source à "tous les autres". Ainsi, pour obtenir tous les PCC entre tous les points , il n'est pas nécessaire d'exécuter n^2 fois l'algorithme mais seulement n fois.

Calcul des PCC

Voici la version **récursive** du calcul de tous les chemins entre deux sommets (appelés *source* et *destination*).

Il faut connaître la liste des chemins entre ces deux sommets et le chemin courant. Pour chaque sommet, on dispose d'une liste d'arcs (appelée "*liste*" ci-dessous) qui permettent d'accéder aux sommets de même potentiel.

L'initialisation de *construireChemin* se fait avec la *source*, la *destination*, un *ensemble vide* et un *chemin vide*.

Algorithme 24 : construireCheminRécursif(source, courant, ensemble, chemin)

```
⇝ test d'arrêt
si (source=courant) alors arrêter

pour i de (courant.liste.taille-1) à 1 pas −1 faire
    chemin' ← copie(chemin)
    chemin' ← courant.liste[i] ∪ chemin'
    ensemble ← ensemble ∪ chemin'
    construireCheminRécursif(source, courant, ensemble, chemin')
fait

chemin ← courant.liste[0] ∪ chemin
construireCheminRécursif(source, courant.liste[0].origine, ensemble, chemin)
```

La version non récursive utilise une pile de chemins à explorer.

Algorithme 25 : construireCheminNonRécursif(source, destination)

```
⇝ initialisations
empiler(chemin_vide), ensemble ← ∅
⇝ calcul des chemins
tant que (pile non vide) faire
    chemin ← dépiler
    sommet ← (chemin=chemin_vide) ? destination : chemin.origine
    si (sommet=origin) alors ensemble ← ensemble ∪ chemin
    sinon
        pour i de (sommet.liste.taille-1) à 1 pas −1 faire
            chemin' ← copie(chemin)
            chemin' ← sommet.liste[i] ∪ chemin'
            empiler(chemin')
        fait
        chemin ← sommet.liste[0] ∪ chemin
        empiler(chemin)
    fin si
fin tant que
retourner ensemble
```

Pour les deux algorithmes, la liste des PCC est donnée par *ensemble*.

Complexité Avec la deuxième version de l'algorithme, on peut évaluer le nombre d'opérations nécessaires pour calculer ces PCC. Le nombre d'opération est égal à

$$\sum_{chemin \in PCC} \Big(\sum \text{nb d'arcs incidents de pcc} \Big)$$

Ce nombre est inférieur à la somme du nombre d'arcs pour tous les chemins de PCC entre la source et la destination. On peut aussi majorer le nombre d'arcs de pcc incident par le nombre d'arcs incidents.

A.7.5. Algorithme de PCC bicritère ou PCC sous contrainte

Il est très fréquent dans les problèmes de réseaux que l'on veuille calculer des plus courts chemins entre deux points, le plus souvent suivant deux objectifs exclusifs (trouver un temps de parcours minimal et de coût minimal). Il faut alors faire appel aux algorithmes de PCC multi-objectifs ou aux algorithmes de calcul du PCC sous contrainte.

Si le problème du calcul de PCC est très facile, le problème des PCC multi-objectif est NP-difficile [CCC99].

A.8. Algorithmes sur les flots

Au cours de la thèse, nous avons eu besoin de déterminer un flot réalisable. Pour cela, nous avons exploité le lemme de Minty décrit dans ce chapitre.

A.8.1. Déterminer un cycle ou un cocyle avec le lemme de Minty

Le lemme de Minty permet de savoir si un arc donné appartient à un cycle ou bien au contraire à un cocycle.

Algorithme 26 : Algorithme de Minty

Entrée : Arc u=(x,y)
$A \leftarrow \{y\}$, $B \leftarrow \{y\}$
$p_y \leftarrow \emptyset$
tant que $B \neq \emptyset$ et $x \notin A$ **faire**
  choisir $n \in B$
  $B = B \backslash \{n\}$
  pour tout $a = (n, z)$, a est vert ou noir et $z \notin A$ **faire**
    $A \leftarrow A \cup \{z\}$, $B \leftarrow B \cup \{z\}$
    $p_z \leftarrow a$
  fait
  pour tout $a = (z, b)$, a est vert ou bleu et $z \notin A$ **faire**
    $A \leftarrow A \cup \{z\}$, $B \leftarrow B \cup \{z\}$
    $p_z \leftarrow a$
  fait
fin tant que
si $x \notin A$ **alors** construire le cocyle associé
sinon construire le cycle associé

A.8.2. Construire un flot réalisable

On détermine un flot réalisable en identifiant des cycles augmentants, c'est-à-dire en cherchant la coloration des arcs du graphe. La couleur d'un arc dépend de la valeur du flot porté par cet arc. Un arc dont la valeur du flot ne peut changer est **rouge** (un tel arc est inintéressant). Un arc qui vérifie la contrainte de capacité supérieure est **noir**, et un arc qui vérifie la contrainte de capacité inférieure est **bleu**. Un arc qui vérifie les deux contraintes est **vert**.

Algorithme 27 : Déterminer un flot réalisable

```
Initialiser toutes les composantes flot du graphe à 0
b ← faux
tant que non (b) faire
    b ← vrai
    pour tout arc u incompatible faire
        rechercher un cycle augmentant
        augmenter le flot sur ce cycle autant que faire se peut
        b ← faux
    fait
fin tant que
```

Annexe B

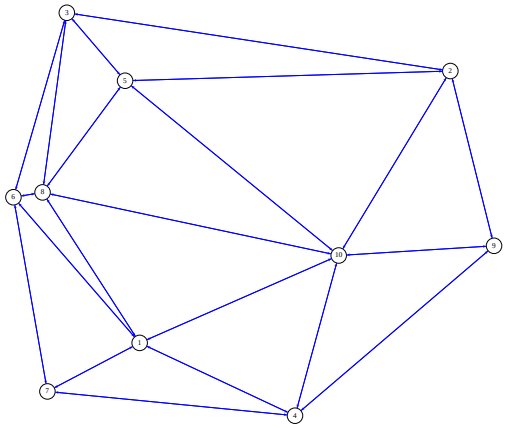
Implémentation & tests

Ce chapitre est dédié à l'implémentation et à la présentation des instances de tests. On trouvera tout d'abord une **présentation détaillées des instances de test** : le graphe support et les différents jeux de commodités. On donnera aussi **quelques exemples de réseaux** trouvés dans la littérature. Dans un deuxième temps, le chapitre reprend quelques points d'implémentation : présentation de la **structure de graphe** utlisée avec quelques définitions de modélisation objet, la **géodésique** et l'utilisation du solveur **Ilog CPLEX**.

Nous ne développons pas ici les techniques utilisées pour une programmation objet efficace. Un chapitre d'annexe, le chapitre D, traite de cette problématique.

B.1. Instances à 10 sommets

Ce graphe triangulé a été généré aléatoirement.



Nombre de sommets	10
Nombre d'arcs	42
Plus petit arc	6083
Plus grand arc	79906
Somme des arcs	
Diamètre	110666
Enveloppe convexe	302334

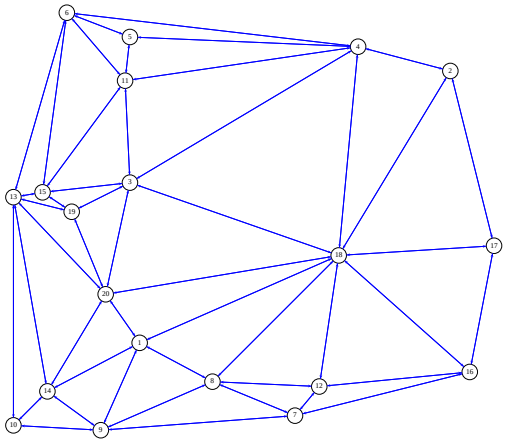
Nous disposons de deux séries de commodités pour ce graphe. Toutes les demandes sont égales avec $D^k = 1, \forall k \in K$. Voici les différentes commodités :

- 30 commodités : on considère les trajets deux à deux (aller-retour) sur l'enveloppe convexe. Six sommets sont concernés.

- 90 commodités : le réseau associé est complet quant au nombre de commodités, de tous les sommets à tous les autres.

B.2. Instances à 20 sommets

Ce graphe, triangulé, a été généré aléatoirement.



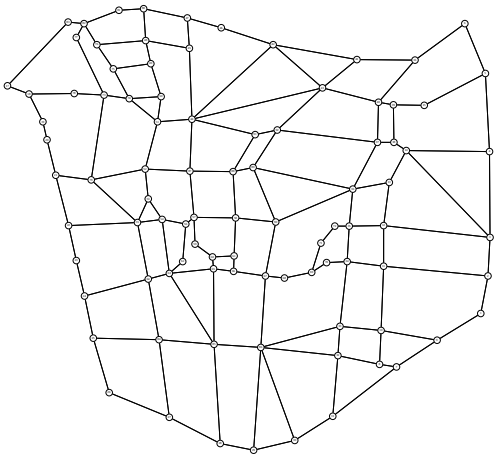
Nombre de sommets	20
Nombre d'arcs	96
Plus petit arc	6083
Plus grand arc	60407
Somme des arcs	2703750
Diamètre	121129
Enveloppe convexe	325446

Nous disposons de deux séries de commodités pour ce graphe. Toutes les demandes sont égales avec $D^k = 1, \forall k \in K$. Voici les différentes commodités :

- 72 commodités : on considère les trajets deux à deux (aller-retour) sur l'enveloppe convexe. Neuf sommets sont concernés.
- 280 commodités : le réseau associé est complet quant au nombre de commodités, de tous les sommets à tous les autres.

B.3. Instances à 100 sommets

Ce graphe, triangulé, est inspiré du centre-ville de Clermont-Ferrand.



Nombre de sommets	96
Nombre d'arcs	304
Plus petit arc	20616
Plus grand arc	261222
Somme des arcs	28252100
Diamètre	1225610
Enveloppe convexe	1000060

Là encore, nous disposons de deux séries de commodités pour ce graphe. Toutes les demandes sont égales avec $D^k = 1, \forall k \in K$. Voici les différentes commodités :

- 12 commodités : on considère les trajets deux à deux (aller-retour) sur l'enveloppe convexe. 4 sommets sont concernés.
- 992 commodités : on considère les trajets deux à deux (aller-retour) sur l'enveloppe convexe. 32 sommets sont concernés.

B.4. Instances de la littérature

Lors de notre recherche bibliographique sur les problèmes de Tournées de Véhicules, nous avons relevé les caractéristiques des différentes instances de tests utilisés. Nous les donnons ici, pour juger de la taille de réseaux "réalistes".

- Ville de Parme : 1134 noeuds, 3016 arcs, 459 stops et 22 lignes [BCC02]
- Twin Cities : 132 lignes, 940 bus et 5 dépôts [DES02]
- Montreal : 206 lignes, 1500 bus et 7 dépôts [DES02]
- Paris : 246 lignes, 3860 bus et 23 dépôts [DES02]
- New-York : 298 lignes, 4860 bus et 18 dépôts [DES02]
- Clermont-Ferrand : 19 lignes (242km), 182 bus (moyenne : 18.1 km/h) [T2C]

B.5. Une structure de graphe générique et efficace

La structure de graphe utilisée a été développée par Bertsekas [BT88, BER89]. Cette structure est particulièrement adaptée pour les graphes dits *statiques*, i.e. ceux dont la structure ne varie pas au cours de l'exécution, et pour le parcours des cocycles des sommets.

Bertsekas a développé un modèle pour les langages procéduraux. Nous donnons la modélisation objet à la figure B.1.

Notre implémentation utilise de manière intensive, non pas le polymorphisme et l'héritage de classe qui sont sources de lenteur pendant l'exécution, mais bien au contraire, les **patrons** de classe (*template* en anglais), encore appelés **classes paramétrées**. Nous discutons à l'annexe D de la pertinence de ce propos. Chaque classe (Nœud, Arc, Chemin) est paramétrée sur le type de données qu'elle peut contenir.

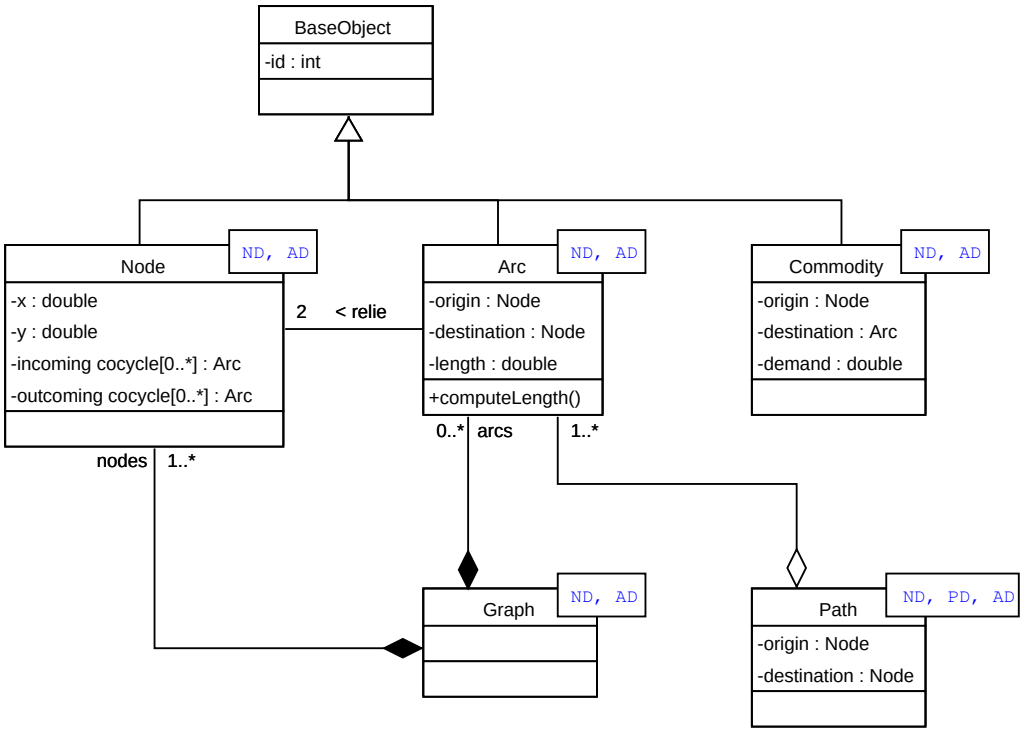


FIG. B.1 – Modèle UML simplifié de la structure de graphe de Bertsekas

Notre implémentation utilise une librairie standard de composants génériques, appelée *Standard Template Library (STL)* (cf [SGI]). Cette librairie propose notamment une interface et une implémentation pour des conteneurs usuels et quelques algorithmes.

Pour garantir la généricité et la réutilisabilité des algorithmes, ceux-ci sont le plus souvent paramétrés par des **foncteur**s, c'est-à-dire une encapsulation d'un fonction au sein d'une classe objet dédiée. C'est une forme limitée du patron de conception *stratégie*.

Nous ne développons pas ici les techniques utilisées pour une programmation objet efficace. Un chapitre d'annexe, le chapitre D, traite de cette problématique.

B.6. Géodésique

Nous donnons ici un modèle simplifié de la géodésique utilisée dans les différentes méta-heuristiques implémentées.

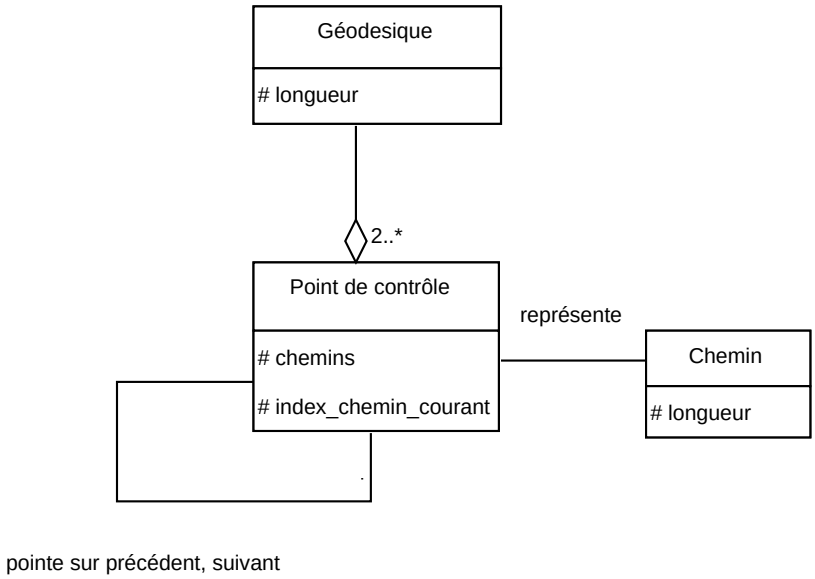


FIG. B.2 – Modèle UML simplifié de la géodésique

B.7. Ilog CPLEX

Pour la thèse, nous avons résolu certains programmes linéaires mixtes avec *Ilog CPLEX* (cf [CPX02a, CPX02b]). Il est possible de l'utiliser suivant deux modes : en ligne de commande ou au travers d'interfaces de communication. Les interfaces de communication sont écrites pour quelques langages usuels (C, C++, Fortran, JAVA ...). Nous nous sommes intéressés à la *Callable Library* pour des programmes écrits en C et à la *Concert Technology* pour les langages objets.

Tous nos programmes ont été écrits en langage objet, il semblait donc naturel d'utiliser la *Concert Technology*. Pourtant, sauf pour quelques tests isolés, nous avons du nous rabattre sur l'interface classique C.

En effet, les expérimentations numériques ont tout d'abord été réalisés sur une machine dont le compilateur C++ natif, CC, est trop ancien pour interpréter correctement la généricité, et sur cette machine, l'intégration *Concert Technology*/*g++* (le compilateur GNU C++) n'est pas possible (Même les pseudo-codes assembleur générés par CC et G++ sont différents). Il est à noter que l'interface C est presque toujours intégrable dans un programme écrit en C++. Nous n'avons détecté qu'une incompatibilité avec l'interfaçage d'un programme écrit et compilé pour un GNU G++ et CPLEX écrit pour MSVC7.0 (i.e. le compilateur .NET de Microsoft) : la librairie dynamique exporte de manière exotique les symboles et gère les types de retour avec légèreté. Les mécanismes d'appel de fonctions diffèrent et ainsi, il n'est

en autre pas possible de manipuler ni les flux CPLEX, ni les fonctions *callback*, c'est-a-dire que l'on s'est privé de fonctionnalités fondamentales comme la surcharge (ou le complément) des fonctions CPLEX natives. Deux solutions s'offraient alors : importer avec précaution les symboles de la librairies ou alors écrire un petit wrappeur C, compilé nécessairement avec G++.

L'autre raison est que la librairie C++ CPLEX était encore en développement lorsque les expérimentations numériques ont été menées sur les modèles exacts. De nombreuses fonctionnalités, que l'on trouve dans la librairie C, ne sont pas encore disponibles (même si la version C++ en offre de nouvelles comme le calcul du temps d'exécution).

Annexe C

Enrichissement des solutions obtenues par optimisation par un couplage avec la simulation

Sommaire

Ce chapitre d'annexe est consacré à la simulation. Il est le fruit d'une collaboration avec Bruno BACHELET et a fait l'objet de plusieurs exposés de conférences et publications.

Sans perte de généralité sur les réseaux de mobilité, nous nous sommes placé dans le cadre où l'on propose un système de transports par navette à des usagers. La demande est élastique. Le modèle d'optimisation permet de prendre en compte les temps d'attente dans le système. Ces temps d'attente sont déterminés par la simulation.

Un tel sujet est très vaste aussi nous avons fortement limité le cadre de notre étude à une simulation par événements discrets. Nous n'avons pas utilisé la notion d'agents pour modéliser le comportement des usagers sur le réseau, comportement dicté par l'optimisation.

C.1. Introduction

Le but de ce chapitre est de proposer un moyen d'améliorer la qualité pratique d'une solution obtenue par une méthode d'optimisation. De nombreuses techniques avancées (voir par exemple la section 1.2 page 10 de ce manuscrit) permettent de résoudre efficacement des problèmes exprimés avec un formalisme mathématique. Des résultats fondamentaux ont été énoncés pour démontrer l'optimalité ou quantifier la qualité d'une solution (les algorithmes d'approximation permettent de trouver une solution dont la qualité par rapport à l'optimum est connue [HOC97]) et de mesurer l'efficacité des méthodes employées (complexité, vitesse de convergence).

Cependant, ces méthodes souffrent d'inconvénients majeurs lorsque l'on cherche à les implémenter sur des cas pratiques. Tout d'abord, ils ne sont pas robustes aux changements de structure du problème : ajouter, par exemple, une nouvelle famille de contraintes peut rendre caduque la méthode d'optimisation utilisée et oblige à en changer (c'est ce qui se passe lorsque l'on résout un problème linéaire avec l'algorithme du simplexe et que l'on ajoute des contraintes non linéaires). Deuxièmement, et il s'agit de notre problématique, il est souvent nécessaire de ne considérer que des modèles simplifiés du problème que l'on cherche à résoudre. Ainsi, une solution optimale pour le problème théorique ne l'est pas forcément pour le problème pratique.

Pour pallier ces défauts, nous proposons une autre technique de couplage simulation/optimisation que l'on a baptisée **enrichissement de modèle**, qui essaie de renforcer le modèle mathématique pour rendre la solution plus adaptée au problème pratique que la solution obtenue par l'optimisation directe. La section 2 rappelle les principes du schéma du couplage classique optimisation-simulation et formalise le problème que l'on aborde dans ce chapitre. La section 3 détaille ce qu'implique l'optimisation directe sur la formulation du problème et les solutions obtenues. La section 4 présente l'enrichissement de modèle, sa finalité et l'algorithme de principe correspondant. Cette proposition est illustrée à la section 5 au travers d'un problème de routage. Les différentes approches présentées dans ce chapitre ont été implémentées pour ce problème et nous comparons la qualité de leurs solutions ainsi que leur efficacité.

C.2. Optimisation-simulation, le couplage classique

On peut exprimer un problème d'optimisation comme la recherche de la meilleure solution x d'un problème réel (P_r), par exemple, comme la minimisation ou la maximisation d'une fonction $f_r(x)$. Une solution x est une solution du problème (P_r) si elle vérifie un ensemble de contraintes qui définit l'espace C_r des solutions réalisables.

$$(P_r) \begin{cases} \textit{optimiser} & f_r(x) \\ \\ \textit{sous les contraintes} & x \in C_r \end{cases}$$

Lors de la phase de modélisation, le problème (P_r) peut être approché. La formulation usuelle de l'optimisation-simulation est donnée par le problème (P_s). Le fait qu'une fonction a (respectivement un espace des solutions) est approchée par une autre fonction b (respectivement un autre espace des solutions) est noté $a \sim b$.

$$(P_s)\begin{cases} \textit{optimiser} & f_s(x) = g(x,\lambda),\ f_s \sim f_r \\ \textit{sous les contraintes} & \left.\begin{array}{r} x \in C_s \\ \lambda \in \Lambda_s(x) \end{array}\right\} \sim C_r \end{cases}$$

C_s représente les contraintes sur la solution x. Cela permet généralement de définir la structure de base d'une solution réalisable (par exemple, le fait qu'une solution soit un cycle dans un graphe orienté). λ est un vecteur de mesures qui est retourné par la simulation pour une solution x (cf figure C.1). $\Lambda_s(x)$ est l'ensemble des solutions réalisables pour λ, relativement à une solution donnée x. Il est défini par les contraintes implicites du modèle de simulation (étant donné une solution x, la simulation retourne l'estimation λ), et par les contraintes explicites sur certaines mesures de performance (par exemple, le délai estimé ne doit pas excéder une limite donnée, ou alors plus simplement, la solution est acceptable). Cela signifie que $\Lambda_s(x)$ contient une solution (x est réalisable si l'on se réfère à l'évaluation par la simulation) ou bien est vide (x n'est pas réalisable suivant l'évaluation par la simulation).

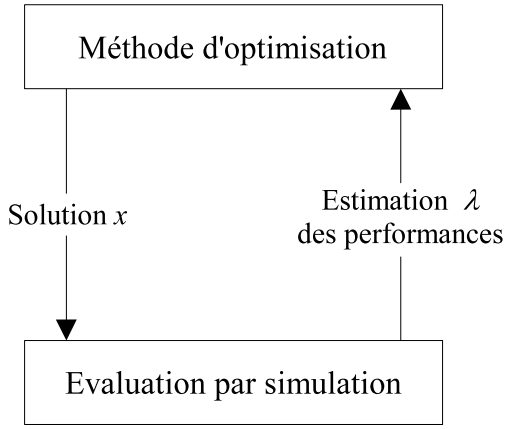


FIG. C.1 – Principe du couplage classique optimisation-simulation

La fonction objectif f_r du problème réel est approchée par la fonction f_s. Nous proposons de paramétrer cette fonction sur le vecteur λ : $f_s(x) = g(x,\lambda)$. Par exemple, λ peut représenter les délais estimés pour une solution x et $g(x,\lambda)$ peut représenter une estimation de la qualité de service de la solution x relativement à ces délais.

L'optimisation-simulation explore l'ensemble des solutions C_s pour optimiser f_s. Pour chaque solution x, la simulation estime le vecteur λ. Si λ satisfait toutes les contraintes de $\Lambda_s(x)$, la solution x est acceptée et la fonction objectif $g(x,\lambda)$ est évaluée.

Plusieurs méthodes sont proposées pour résoudre le problème (P_s). Une classification en quatre approches majeures peut être trouvée dans [MV94] et [AZA92] : les méthodes de gradient, l'approximation stochastique, la surface de réponse et les heuristiques. Toutes ces méthodes sont robustes au changement de fonction objectif ou aux modifications de contraintes du problème. Cependant, elles ne représentent qu'une partie des techniques d'optimisation et leur efficacité ainsi que leur convergence ne sont pas toujours assurées.

C.3. Optimisation directe

Les techniques d'optimisation, indépendantes de la simulation, nécessitent généralement l'emploi de modèles mathématiques. Cette représentation est aussi une approximation du problème réel (P_r).

$$(P_o) \begin{cases} \textit{optimiser} & f_o(x),\ f_o \sim f_r \\ \\ \textit{sous les contraintes} & x \in C_o, C_o \sim C_r \end{cases}$$

Cependant, on peut raisonnablement supposer que le problème d'optimisation-simulation (P_s) est plus proche du problème réel que ne l'est le problème par optimisation directe (P_o). Pour présenter l'enrichissement de modèle, nous allons faire à partir de maintenant quelques hypothèses sur la structure de (P_o).

$$(P_o) \begin{cases} \textit{optimiser} & f_o(x) = g(x, \lambda) \\ \\ \textit{sous les contraintes} & x \in C_o \supset C_s \\ & \lambda \in \Lambda_o(x),\ \Lambda_o \sim \Lambda_s \end{cases}$$

- Les contraintes C_s décrivent la structure de base de la solution réalisable. De plus, nous pouvons raisonnablement supposer que les contraintes définissant C_o sont des approximations, voire même des relaxations, des contraintes définissant C_s. Cela signifie que l'on a l'inclusion $C_o \supset C_s$.
- Nous considérons que les fonctions objectif f_o et f_s sont identiques : $f_o(x) = g(x, \lambda)$. Cependant, nous supposons que l'espace des solutions Λ_o est une approximation de l'espace des solutions Λ_s, ce qui implique naturellement que f_o est une approximation de f_s.

Suivant la structure du modèle, de nombreuses techniques d'optimisation peuvent être utilisées pour résoudre le problème (P_o). Cependant, une fois qu'une méthode particulière a été choisie pour résoudre le problème, il est difficile de gérer les changements sur le type de contraintes du problème. Pour l'enrichissement de modèle, nous ne considérons que des changements sur les contraintes qui définissent Λ_o. On appelle K_m l'ensemble de toutes les familles de contraintes qui sont supportées par la méthode m. Cela signifie que le problème (P_o) peut être résolu par la méthode m si et seulement si $\Lambda_o \in K_m$.

C.4. Enrichissement de modèle

Avec toutes les hypothèses précédentes, si $\Lambda_o(x)$ n'est jamais vide, toutes les solutions $x \in C_o$ sont des solutions réalisables du problème (P_o). Comme $C_o \supset C_s$, toute solution du problème d'optimisation-simulation (P_s) est une solution réalisable de (P_o). En particulier, toute solution optimale x_s^* de (P_s) est une solution réalisable de (P_o). L'idée qui se cache derrière l'enrichissement de modèle est de trouver une famille de contraintes $\Lambda_o \in K_m$ telle que la solution optimale x_o^* de (P_o) corresponde à une solution optimale x_s^* de (P_s). Nous

pouvons formaliser l'enrichissement de modèle comme suit :

$$(P_e)\begin{cases} \textit{optimiser} & f_s(x_o^*) = g(x_o^*, \lambda_s) \\ \textit{sous les contraintes} & \Lambda_o \in K_m, \Lambda_o \sim \Lambda_s \\ & (x_o^*; \lambda_o^*) \textit{ solution optimale de } (P_o) \\ & \lambda_s \in \Lambda_s(x_o^*) \end{cases}$$

(P_e) est un problème vraiment très difficile. Cependant, un moyen de trouver une bonne solution à ce problème peut être d'approximer Λ_s par Λ_o aussi précisément que possible. Ainsi, λ_s et λ_o^* auront des valeurs similaires et l'évaluation théorique $g(x_o^*, \lambda_o^*)$ de la solution optimale x_o^* du problème (P_o) sera proche de celle obtenue par l'évaluation de la simulation $g(x_o^*, \lambda_s)$.

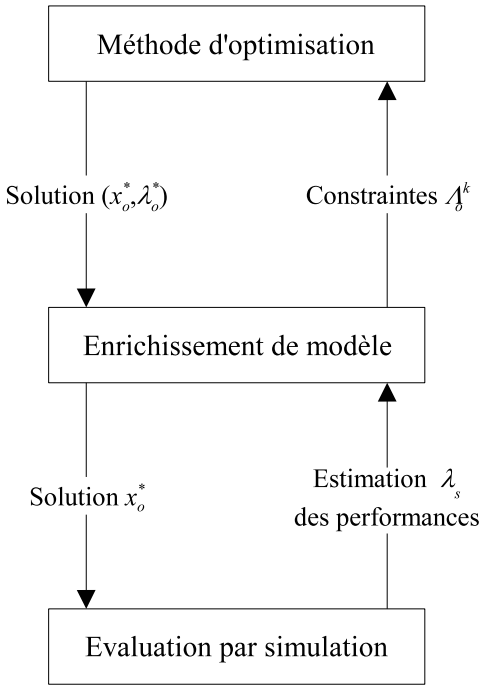


FIG. C.2 – Heuristique de l'enrichissement de modèle

L'algorithme 28 et la figure C.2 décrivent l'approche heuristique de l'enrichissement de modèle qui modifie itérativement Λ_o, en s'appuyant sur l'évaluation de la simulation λ_s d'une solution optimale de (P_o).

Algorithme 28 : Heuristique fondée sur l'enrichissement de modèle

$k \leftarrow 0$
soit n le nombre maximal d'itérations
soit Λ_o^k une approximation de Λ_s
répéter
  soit (x_o^*, λ_o^*) une solution optimale de (P_o) avec $\Lambda_o = \Lambda_o^k$
  soit λ_s l'évaluation par simualtion de x_o^*
  $\Lambda_o^{k+1} \leftarrow h(\Lambda_o^k, \lambda_s)$
  $k \leftarrow k + 1$
jusqu'à $|g(x_o^*, \lambda_o^*) - g(x_o^*, \lambda_s)| < \varepsilon$ OU $k \geq n$

La fonction h représente la manière de modifier Λ_0 à chaque itération. Cette fonction est spécifique au problème considéré. Dans la prochaine section, nous allons illustrer cette heuristique avec un problème de routage.

C.5. Application à un problème de routage

Nous proposons d'étudier un problème de routage pour illustrer le propos des sections précédentes. Nous allons détailler la méthode d'optimisation-simulation, l'approche par optimisation directe et l'enrichissement de modèle pour ce problème particulier. Nous présentons aussi une comparaison pratique de ces différentes approches

C.5.1. Présentation du problème

Nous allons nous intéresser spécifiquement au problème suivant : on considère une compagnie de transports publics urbains, une compagnie de bus par exemple. Cette compagnie a besoin de concevoir un système de transport dont le coût est contrôlé et qui satisfait des utilisateurs potentiels. Il s'agit donc d'une application des réseaux de mobilité.

Nous allons utiliser les notations introduites au chapitre 2 de cette thèse. Considérons un réseau orienté $G = (V, E)$ où V est l'ensemble des nœuds et E l'ensemble des arcs. Chaque sommet représente un arrêt de bus potentiel (point d'entrée ou de sortie sur le réseau) ou un croisement dans le réseau réel. Chaque arc représente un morceau de rue entre deux stops ou croisements.

Nous supposons que les demandes des usagers sont connues, c'est-à-dire que nous disposons d'une description des mouvements que les usagers souhaitent réaliser sur le réseau. La commodité k (parmi un ensemble K) regroupe les données suivantes : $o_k, d_k, D^k, t^k_{\min}, t^k_{\max}$, respectivement l'origine, la destination, la demande nominale, un temps de référence minimal, un temps de référence maximal de la commodité. $t^k_{\max}$ représente par exemple le temps de marche à pied nécessaire pour se rendre de o^k à d^k et $t^k_{\min}$ le temps pour effectuer le même trajet avec un véhicule personnel ou un bus. Les usagers ont un comportement "paresseux", c'est-à-dire qu'ils empruntent le système de transport au plus près et en descendent au plus près.

Par la suite, nous nous limitons au problème de trouver un système de transport Γ qui respecte les contraintes suivantes :

- Γ est un ensemble de circuits dans G (pour plus de simplicité, on se limitera à un seul circuit pour la suite) ;

- la taille des tournées de bus, c'est-à-dire le temps mis pour décrire le circuit, doit être inférieure à une borne donnée ;
- Γ doit maximiser la satisfaction des usagers.

Notons t^k le temps induit par Γ pour effectuer le trajet de la commodité k. La satisfaction peut être modélisée par une fonction Φ^k définie comme suit : si $t^k \leq 2t^k_{\min}$, l'usager est pleinement satisfait et $\Phi^k(t^k) = 1$, si au contraire $t^k \geq 2t^k_{\max}$, l'usager n'est plus du tout satisfait et alors $\Phi^k(t^k) = 0$; entre ces deux bornes, plus le temps de parcours est proche du temps le plus court, plus l'usager est satisfait. La figure C.3 illustre la fonction de satisfaction.

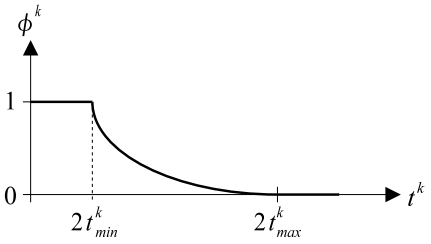


FIG. C.3 – Fonction de satisfaction pour une commodité donnée

Pour finir, nous définissons w_i comme le temps d'attente des usagers à l'arrêt i. Si l'on fait un parallèle avec les notations introduites précédemment, λ est un vecteur constitué de tous les temps de parcours t^k et des temps d'attente w_i ; λ_s est l'estimation en pratique de ces temps (obtenus par simulation).

Appelons x la solution du problème. x_e vaut 1 lorsque l'arc e appartient à la tournée de bus et vaut 0 sinon. La fonction objectif consiste à maximiser la demande des usagers, c'est-à-dire $\sum_k D^k \Phi^k(t^k)$.

C.5.2. Optimisation-simulation

Nous proposons de résoudre le problème en utilisant la métaheuristique tabou introduite au chapitre 3 dans l'algorithme de simulation-optimisation (P_s) avec la structure de géodésique. Ainsi C_s représente l'ensemble des géodésiques dont la longueur respecte la contrainte de taille.

Pour une solution donnée x du problème d'optimisation-simulation (P_s), la simulation permet d'estimer les temps de parcours des différentes commodités avec les temps d'attente w_i. En dehors des contraintes implicites de la simulation, nous n'envisageons pas de contraintes additionnelles dans Λ_s qui puissent éliminer une solution x connaissant son estimation λ. Cela signifie que toute solution $x \in C_s$ est une solution réalisable de (P_s) (c'est une condition nécessaire pour l'enrichissement de modèle fonctionne).

Nous considérons comme graphe de travail un graphe planaire à 109 nœuds et 392 arcs représentant le centre de Clermont-Ferrand (cette instance possède un maillage un peu plus dense que celui utilisé dans le reste de la thèse) avec 12 demandes. Le tableau C.1 présente des résultats numériques pour l'optimisation-simulation sur cette instance. Pour tester différentes structures de problème, nous faisons varier le nombre de points de contrôle de la géodésique. Nous faisons aussi varier la taille maximale de la tournée. Le couplage a été réalisé en C++ et le simulateur avec la plateforme *B++ Simulator*. Les tests ont été réalisés sur un Pentium Centrino 1.7Ghz avec le compilateur G++ 3.2 (avec l'option -O2).

Points de Contrôle	Taille max T	Valeur record $f_s(x_s^*)$	Nombre d'évaluations	Temps CPU (h)
5	700000	10.33	3793	5h04
5	800000	10.30	5203	7h04
5	900000	10.32	4807	6h08
5	1000000	10.32	7573	9h44
7	700000	10.32	8201	10h44
7	800000	10.34	6092	8h16
7	900000	10.32	6749	8h56
7	1000000	10.31	7883	10h28
10	700000	10.32	8843	11h20
10	800000	10.33	9706	10h12
10	900000	10.31	10732	13h52
10	1000000	10.34	8812	11h04
15	700000	10.31	6074	8h20
15	800000	10.32	8759	10h08
15	900000	10.33	10699	11h16
15	1000000	10.33	10302	11h08

TAB. C.1 – Résultats numériques du couplage optimisation-simulation

Les temps d'exécution peuvent être relativement longs et s'étaler sur plusieurs jours. Comme notre but dans ce chapitre est de connaître des bonnes solutions pour les comparer avec celles obtenues par l'enrichissement de modèle, nous avons choisi de limiter le nombre de diversifications de l'algorithme tabou. On peut remarquer que le nombre d'évaluations par simulation pour résoudre (P_s) est élevé, et sans cette restriction, ce nombre peut dépasser les 100000 évaluations. En cela, les solutions données ici ne sont pas toujours les meilleures que l'on pourrait obtenir avec le schéma de résolution non contraint en temps.

C.5.3. Optimisation directe

Comme nous l'avons montré dans le premier chapitre de cette thèse, ce problème de routage peut être considéré à la fois comme un problème de synthèse de réseau ou un problème de tournées de véhicule. Les techniques de résolution que nous avons citées sont donc applicables. Nous avons aussi montré au chapitre 3 que les solutions obtenues avec la métaheuristique tabou étaient très proches, voire égales, aux solutions optimales quand celles-ci sont connues. C'est donc cette méthode de résolution que nous avons retenue ici pour les besoins de comparaison et résoudre (P_o). Avec les progrès récents de développement de solveurs comme Ilog CPLEX, on peut aussi utiliser le formalisme de programmation entière mixte si le temps ou l'espace mémoire n'est pas un problème (sur des problèmes de taille encore modeste). Avec ce formalisme, C_o est plus grand que C_s ($C_s \subset C_o$). Ce n'est pas le cas avec la métaheuristique tabou où l'on n'utilise exclusivement comme description de tournées que la géodésique et non un circuit général (et ainsi $C_o = C_s$).

Λ_o est une approximation de Λ_s. Les contraintes implicites de simulation qui définissent Λ_s sont remplacées par des contraintes qui permettent d'évaluer les temps de parcours t^k induits par le système de transport. D'autres contraintes imposent que les temps d'attente w_i soient nuls dans la mesure où le modèle fait l'hypothèse que le nombre de véhicules est suffisamment important pour que les temps d'attente n'interviennent pas. En résumé, Λ_o est une manière d'estimer les temps de parcours avec des temps d'attente nuls.

Points de contrôle	Taille max T	Optimisation directe Eval. théorique $f_o(x_o^*)$	Optimisation directe Eval. Simulation $f_s(x_o^*)$	Couplage classique Meilleure solution $f_s(x_s^*)$	Temps CPU (s)
5	700000	11.99	10.29 (-0.4%)	10.33	5.0
5	800000	11.99	10.25 (-0.5%)	10.30	5.0
5	900000	11.99	9.62 (-6.8%)	10.32	5.1
5	1000000	11.99	10.08 (-2.4%)	10.32	5.3
7	700000	11.99	10.31 (-0.1%)	10.32	6.8
7	800000	11.99	9.92 (-4.1%)	10.34	7.1
7	900000	11.98	9.79 (-5.1%)	10.32	7.5
7	1000000	12.00	9.43 (-8.5%)	10.31	7.6
10	700000	11.97	10.18 (-1.4%)	10.32	7.7
10	800000	11.99	9.92 (-4.0%)	10.33	8.9
10	900000	11.99	10.05 (-2.5%)	10.31	9.4
10	1000000	12.00	9.33 (-9.8%)	10.34	10.3
15	700000	11.99	10.29 (-0.2%)	10.31	8.0
15	800000	11.99	10.01 (-3.0%)	10.32	9.0
15	900000	11.99	9.85 (-4.6%)	10.33	9.8
15	1000000	12.00	9.56 (-7.5%)	10.33	10.8

TAB. C.2 – Résultats numériques de l'optimisation directe

Nous avons réalisé les tests pour (P_o) dans les mêmes conditions que les tests pour (P_s). Le problème est résolu par la méthode tabou. Le tableau C.2 donne l'évaluation théorique $f_o(x_o^*)$ des meilleures solutions x_o^* obtenues pour (P_o) et leur évaluation avec la simulation $f_s(x_o^*)$. La résolution est relativement rapide (une dizaine de secondes environ) comparée aux temps de résolution précédents (plusieurs heures) mais les solutions ne donnent pas toujours de bonnes évaluations par la simulation. On indique entre parenthèses la différence relative $(f_s(x_o^*) - f_s(x_s^*))/f_s(x_s^*)$.

C.5.4. Enrichissement de modèle

L'optimisation-simulation donne de bons résultats pratiques, mais requiert un temps d'exécution très long. A l'opposé, l'optimisation directe est très rapide mais donne des solutions pratiques qui ne sont pas toujours très bonnes. Nous allons mettre en œuvre maintenant l'enrichissement de modèle sur le problème de routage pour affiner les contraintes qui définissent Λ_o et ainsi obtenir de meilleures solutions pratiques pour (P_o).

Pour résoudre le problème (P_o), nous supposons que les temps d'attente sont nuls. Ce qui peut faire que Λ_o est une approximation de piètre qualité de Λ_s. Comme la méthode utilisée pour résoudre (P_o) permet de prendre en compte des temps d'attente non nuls, nous proposons de modifier les temps d'attente théoriques à chaque itération de l'enrichissement de modèle (et ainsi affiner l'approximation $\Lambda_o$de Λ_s).

Appelons w_i^j les temps d'attente du problème (P_o) à l'itération j, et $\lambda_s = (w\ t)$, composé des vecteurs $w = (w_i)_{i \in V}$ (les temps d'attente estimés) et $t = (t_k)_{k \in K}$ (les temps de trajet estimés), l'évaluation par simulation de la solution théorique courante x_o^*. Si $w_i \neq 0$, cela signifie que le sommet i est effectivement utilisé comme arrêt de bus. Nous proposons de faire tendre w_i^{j+1} vers w_i comme suit : $w_i^{j+1} \leftarrow w_i^j + \Delta(w_i - w_i^j)$. Si $w_i = 0$, cela signifie que le sommet i n'est pas un arrêt auquel montent les usagers. Cependant, nous allons modifier le temps d'attente correspondant w_i^{j+1} pour qu'il tende vers la moyenne des temps d'attente

M (temps estimés depuis le tout début de l'algorithme) de la manière suivante : $w_i^{j+1} \leftarrow w_i^j + \Delta(M - w_i^j)$.

$\Delta < 1$ est un pas de progression que l'on doit ajuster. On peut le choisir soit constant, soit comme une variable dépendante du nombre d'itérations (pour assurer une certaine convergence). Pour le problème de routage considéré, nous avons décidé de fixer $\Delta = 0.1$ et nous arrêtons l'algorithme de l'enrichissement de modèle après $n = 100$ itérations, ou alors lorsque l'algorithme converge, en d'autres termes lorsque la différence entre l'évaluation théorique $f_o(x_o^*) = g(x_o^*, \lambda_o^*)$ te l'évaluation par la simulation $f_s(x_o^*) = g(x_o^*, \lambda_s)$ est plus petite qu'un seuil $\varepsilon = 0.01$. L'algorithme 29 résume l'heuristique de l'enrichissement de modèle pour notre problème de routage.

Nous avons réalisé les tests pour l'enrichissement de modèle dans les mêmes conditions que pour les problèmes (P_s) et (P_o). Le tableau C.3 donne l'évaluation théorique des meilleures solutions x_e^* trouvées. Cela montre que cette évaluation est finalement très proche de celle par simulation. Entre parenthèses, nous mesurons l'amélioration relative de la qualité de la solution pratique avec elle obtenue par l'optimisation directe $(f_s(x_e^*) - f_o(x_o^*))/f_s(x_o^*)$.Nous avons aussi relevé le nombre d'itérations effectivement réalisées.

Algorithme 29 : Heuristique de l'enrichissement de modèle appliquée à un problème de routage

$j \leftarrow 0$
pour chaque sommet $i \in V$ **faire** $w_i^j \leftarrow 0$
répéter
  $j \leftarrow j + 1$
  Résoudre (P_o) avec les temps d'attente w_i^j, $i \in V$
  soit (x_o^*, λ_o^*) une solution optimale de (P_o)
  soit $\lambda_s = (w\ t)$ l'évaluation par simulation de x_o^*
  mettre à jour la moyenne M des temps d'attente estimés
  si $w_i \neq 0$ **alors** $w_i^{j+1} \leftarrow w_i^j + \Delta(w_i - w_i^j)$ **sinon** $w_i^{j+1} \leftarrow w_i^j + \Delta(M - w_i^j)$
jusqu'à $|g(x_o^*, \lambda_o^*) - g(x_o^*, \lambda_s)| < \varepsilon$ or $j \geq n$

La résolution de l'enrichissement de modèle est plus lente que celle par optimisation directe mais est vraiment plus rapide que celle du couplage classique optimisation-simulation. En fait, on peut remarquer que le nombre d'évaluations par la simulation est relativement faible.ce qui n'est absolument pas le cas avec le schéma de couplage classique. La qualité de la solution obtenue avec l'enrichissement de modèle est vraiment **très proche** de celle obtenue avec le couplage classique et améliore **toujours** celle de l'optimisation directe.

Points de contrôle	Taille max T	Enrich. de modèle Théorique $f_o(x_e^*)$	Simulation $f_s(x_e^*)$	Nb iter.	Opti. directe $f_s(x_o^*)$	Couplage classique $f_s(x_s^*)$	Temps CPU (min)
5	700000	10.48	**10.48** (+1.8%)	51	10.29	10.33	7'46
5	800000	10.38	**10.38** (+1.3%)	51	10.25	10.30	7'47
5	900000	10.20	10.20 (+6.0%)	45	9.62	**10.32**	6'53
5	1000000	10.23	10.23 (+1.5%)	49	10.08	**10.32**	7'20
7	700000	10.49	**10.49** (+1.7%)	46	10.31	10.32	8'24
7	800000	10.40	**10.41** (+4.9%)	46	9.92	10.34	8'33
7	900000	10.21	10.22 (+4.4%)	52	9.79	**10.32**	9'28
7	1000000	10.21	10.21 (+8.3%)	33	9.43	**10.31**	6'09
10	700000	10.47	**10.47** (+2.8%)	50	10.18	10.32	9'08
10	800000	10.41	**10.41** (+4.9%)	33	9.92	10.33	7'01
10	900000	10.12	10.12 (+0.7%)	56	10.05	**10.31**	12'18
10	1000000	10.05	10.05 (+7.7%)	64	9.33	**10.34**	15'00
15	700000	10.48	**10.49** (+1.9%)	51	10.29	10.31	10'02
15	800000	10.42	**10.42** (+4.0%)	44	10.01	10.32	9'17
15	900000	10.22	10.22 (+4.0%)	56	9.85	**10.33**	12'35
15	1000000	10.06	10.06 (+5.2%)	61	9.56	**10.33**	14'41

TAB. C.3 – Résultats numérique de l'enrichissement de modèle

La figure C.4 montre l'évolution et la convergence des deux évaluations, théorique et par simulation, de x_o^* à chaque itération j de l'enrichissement de modèle, dans la recherche d'une 10-géodésique avec contrainte de taille $T = 1000000$.

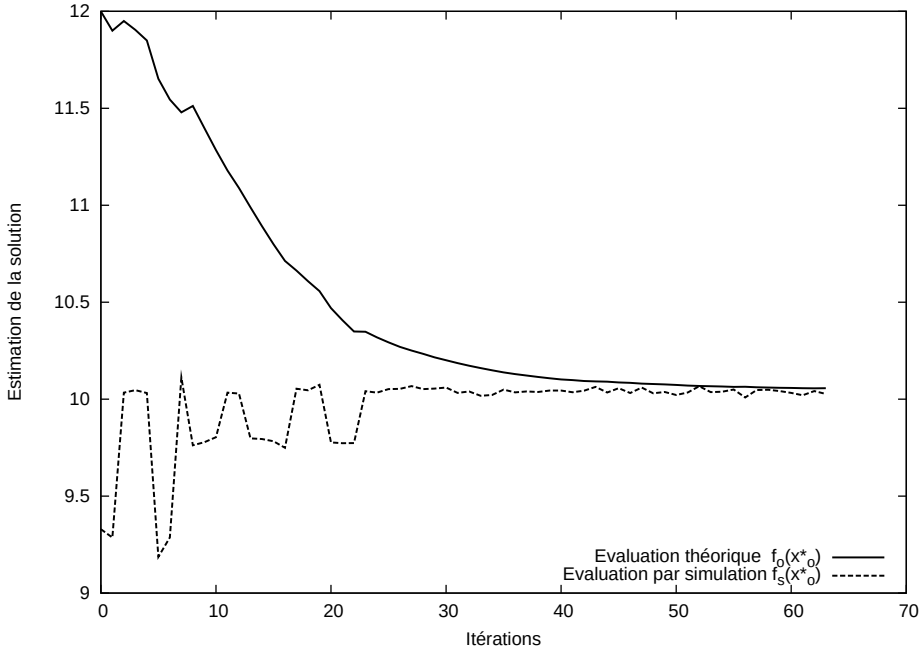


FIG. C.4 – Évolution des solutions de l'enrichissement de modèle (Cas 1)

La convergence semble avoir été facilement obtenue dans ce cas. Cependant, la figure C.5 montre le même type d'évolution dans la recherche d'une 7-géodésique de taille maximale $T = 1000000$. L'heuristique oscille quelques fois entre plusieurs bonnes solutions et la convergence est, de ce fait, plus difficile à obtenir.

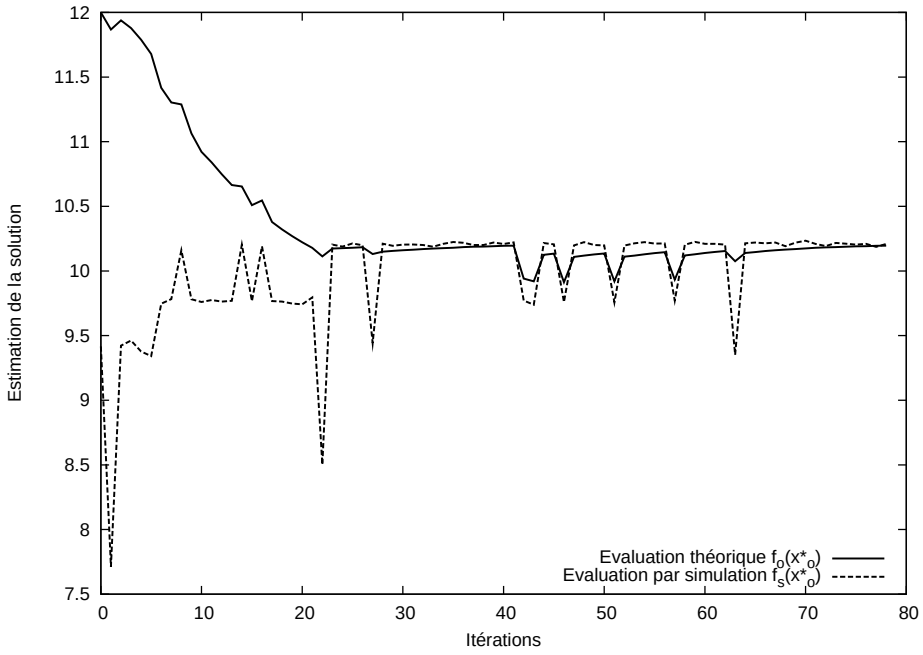


FIG. C.5 – Évolution des solutions de l'enrichissement de modèle (Cas 2)

C.6. Conclusion

Le couplage classique optimisation-simulation donne de bonnes solutions pratiques. Cependant, cela nécessite généralement un temps d'exécution relativement long, avec de trop nombreuses évaluations par simulation. D'un autre côté, certains problèmes, comme le problème de routage considéré ici, peuvent être résolus très efficacement. Malheureusement, les solutions théoriques obtenue de cette manière ne sont pas toujours très bonnes en pratique.

Nous avons proposé dans ce chapitre une approche que l'on a appelée enrichissement de modèle qui reste axée sur la résolution du problème théorique mais qui essaie d'améliorer sa formulation en prenant en compte des paramètres pratiques qui sont évalués grâce à la simulation. Le but avoué de cette approche est de combiner les avantages des différentes méthodes et d'obtenir rapidement de bonnes solutions pratiques.

Nous avons formalisé l'enrichissement de modèle comme le problème (P_e) qui, sous certaines conditions, doit permettre d'obtenir les meilleures solutions pratiques relativement à l'évaluation par simulation. Un tel problème semble être très difficile à résoudre mais permet de considérer le couplage entre l'optimisation et la simulation sous un jour nouveau. Nous avons illustré notre propos sur un problème de routage résolu avec une heuristique inspirée de l'enrichissement de modèle. Les expérimentations montrent le potentiel de l'enrichissement de modèle, une piste qu'il faudrait bien entendu approfondir.

Nous terminons ce chapitre par une copie d'écran de l'interface graphique du simulateur. Le graphe de travail est doublé par le plan de la ville (le plan, très reconnaissable, n'est pas libre de droits et appartient au site dont il est tiré)

FIG. C.6 – Interface graphique du simulateur

Annexe D

Conception d'algorithmes génériques

Sommaire

D.1. Introduction

Ce chapitre est le fruit d'une collaboration avec Bruno BACHELET et Antoine MAHUL et les trois premières sections ont été publiées dans la revue *Software, Practice and Experimence* (Wiley) dans un article intitulé *Designing Generic Algorithms for Operations Research.* La dernière section se rapporte à un problème qui est encore ouvert et que l'on rencontre fréquemment en Recherche Opérationnelle : la maintenance simultanées de différents modèles pour un même objet.

Ce chapitre est né du constat que bien que nos thèses aient été réalisées dans des domaines bien différents (la synchronisation hypermédia [BAC03], l'approximation neuronale dans les réseaux de communication [DMA03] et les problèmes de moblité), nous avons été confronté au même type de challenge : développer des algorithmes génériques qui soient aussi efficaces.

Nous allons donner des solutions de conception à l'instar des **patrons de conception** — (*design pattern*) dans la langue de Shakespeare — c'est-à-dire une description d'un problème et des solutions envisageables pour le résoudre. La description est indépendante d'un quelconque langage de programmation mais peut faire intervenir certaines spécificités.

La recherche opérationnelle utilise des méthodes mathématiques sophistiquées pour optimiser des systèmes. Leur implémentation est souvent difficile et prendre en considération des problématiques de génie logiciel avancées la rend encore plus complexe. En outre, la taille de certains problèmes pratiques implique une implémentation efficace des algorithmes si l'on souhaite obtenir des temps de calcul acceptables. Pour toutes ces raisons, certains praticiens en recherche opérationnelle favorisent la programmation procédurale (C, Fortran...). Le but de ce chapitre n'est pas de présenter de nouveaux concepts de programmation, mais de montrer que concevoir des algorithmes génériques et efficaces pour la recherche opérationnelle (ou tout autre domaine scientifique où les mêmes types de problèmes sont présents) est possible en combinant la programmation par objets et la programmation générique.

Dans ce chapitre, le terme algorithme représente un traitement complexe que l'on cherche à rendre *générique*, c'est-à-dire que l'algorithme doit être extensible (son comportement peut être adapté "simplement" pour répondre à divers objectifs, ce qui est aussi appelé la *réutilisabilité*) et indépendant (bien qu'il puisse interagir fortement avec d'autres composants logiciels, structures de données ou algorithmes, il doit rester le plus indépendant possible de ceux-ci). En résumé, on cherche à concevoir des composants adaptables au plus de situations possibles et interchangeables lorsque leurs fonctions sont similaires. La difficulté est d'atteindre ces objectifs tout en conservant une très bonne efficacité des algorithmes, fondamentale dans des domaines scientifiques comme la recherche opérationnelle.

Diverses raisons peuvent conduire à une conception par objets inefficace, des mauvais choix du concepteur aux limitations du paradigme. Plus spécifiquement, les applications scientifiques ne semblent pas très bien adaptées à un usage important de l'héritage, et malheureusement, lorsqu'on cherche l'extensibilité, ce concept semble incontournable dans une conception par objets. Souvent les algorithmes manipulent de nombreuses données de petite taille (comme les éléments d'une matrice) et le simple fait d'accéder à ces données par une méthode virtuelle peut conduire à de mauvaises performances. Une méthode virtuelle ne peut être liée au code qui l'appelle qu'au moment même de l'appel, par le mécanisme appelé *liaison dynamique* (*dynamic binding*), et nécessite donc plus de temps qu'une méthode classique, liée statiquement, pour être exécutée. Cependant, si l'on regarde par exemple les expériences menées dans [O'R02] et [LIP96] pour le langage C++, on s'aperçoit que cette différence de temps d'exécution entre liaisons statique et dynamique est relativement faible : environ 7 % de surcoût.

En outre, si le langage permet le *déroulement* (*unrolling*) d'une méthode (i.e. l'appel à la méthode est remplacé directement par son corps), alors il est possible d'obtenir un temps d'exécution de la méthode bien inférieur à celui d'une exécution classique. L'expérience de [LIP96] (confirmée par les résultats de [O'R02]) montre un gain d'un facteur 50 de l'exécution déroulée (à noter que ce gain dépend très fortement du contenu de la méthode appelée). Ce gain s'explique principalement par les optimisations que peut réaliser le compilateur seulement dans la version déroulée. Comme la liaison dynamique empêche le déroulement d'une méthode virtuelle, on peut comprendre l'importance d'éviter cette possibilité pour certaines méthodes critiques (i.e. au contenu très rapide d'exécution et appelées très souvent).

Bien que des solutions de conception existent pour tenter d'éviter la liaison dynamique (par exemple, la *délégation* [JZ91]), la notion de généricité proposée actuellement dans de nombreux

langages (Ada, C++, Java, C#...), et qui a donné naissance à la programmation dite *générique* [MS89], est souvent préférée. Notre choix s'est porté sur le langage C++ pour développer les algorithmes, l'un des langages objets disposant de la généricité et d'un mécanisme de déroulement des méthodes. Bien que ce langage ne soit pas des plus riches concernant les concepts de la programmation générique (voir [GJL+03] pour une comparaison détaillée), il permet néanmoins un passage relativement naturel des langages populaires en recherche opérationnelle comme C et Fortran vers la programmation par objets et la programmation générique. Dans ce chapitre, nous limitons notre étude aux algorithmes qui sont exprimés en premier lieu sous forme procédurale, et notamment nous ne considérons pas ceux écrits sous forme fonctionnelle qui méritent très certainement une étude différente.

La section 2 rappelle différentes solutions pour concevoir des structures de données génériques et des algorithmes aussi indépendants que possible de celles-ci. Nous proposons une comparaison de leur efficacité sur l'implémentation d'une structure de graphe en recherche opérationnelle. La section 3 présente des solutions de conception simples qui permettent de rendre les algorithmes fondamentalement plus génériques. Ces solutions sont implémentées sur un algorithme de plus court chemin afin de permettre une comparaison de leur efficacité. La section 4 explique que des algorithmes peuvent avoir besoin d'ajouter des données aux paramètres qu'ils reçoivent : par exemple, la résolution d'un plus court chemin dans un graphe nécessite l'ajout d'un potentiel sur les nœuds. Mais, dans le souci de maintenir l'encapsulation de tels algorithmes, il ne peut pas être envisagé que l'utilisateur de l'algorithme ajoute lui-même ces données supplémentaires. Plusieurs solutions sont discutées pour maintenir la généricité de l'algorithme, et celle retenue est implémentée pour être comparée à une conception plus classique. La section 5 explique que, parfois, un problème peut être modélisé de différentes manières : par exemple un problème de plus court chemin peut être modélisé sur la base d'un graphe ou alors directement par un programme linéaire. Suivant la technique de résolution, ces différents modèles du même problème peuvent être manipulés en même temps. Des solutions sont proposées pour faciliter le passage d'un modèle à un autre, ainsi que l'échange de données entre eux. La section 6 conclut sur l'utilisation des différentes solutions présentées ici dans le cadre de projets de recherche opérationnelle.

D.2. Structures de données génériques

D.2.1. Généricité et héritage

Pour concevoir des structures de données efficaces et génériques, la solution communément employée est une forme de généricité qui ne fournit pas une simple classe mais une classe paramétrable, c'est-à-dire un modèle de classe, pour une structure de données. Elle décrit la classe elle-même et quelques types de données que celle-ci manipule comme paramètres. Ainsi, pour différents jeux de paramètres, le modèle est instancié, fournissant des classes paramétrées, pour lesquelles le processus complet de compilation et d'optimisation est réalisé, de manière complètement équivalente à des classes écrites manuellement. Cela signifie que la généricité dans une conception n'a aucun impact négatif direct sur son efficacité.

Considérons maintenant une classe `Graph` qui représente un graphe orienté en recherche opérationnelle (un graphe est composé de nœuds et d'arcs, où chaque arc relie deux nœuds). L'objectif est de fournir une structure de données unique qui peut être utilisée pour modéliser différentes sortes de graphes, par exemple des graphes de flot qui modélisent des flux circulant d'un point à un autre ou des graphes géographiques qui modélisent des positions avec des

coordonnées et des routes qui les séparent. Cela signifie que la structure de données doit être capable de porter divers types de données à la fois sur les arcs et sur les nœuds du graphe.

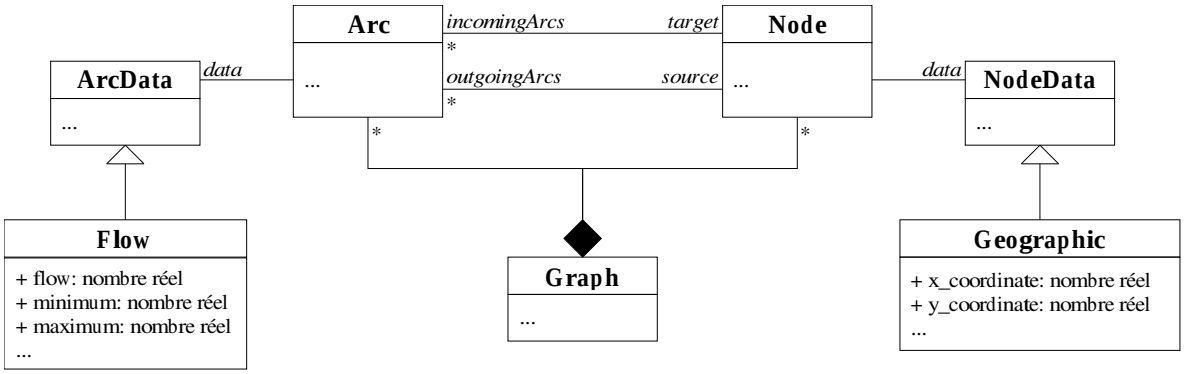


FIG. D.1 – Graphe modélisé par héritage.

Un modèle utilisant l'héritage est proposé à la figure D.1 (dans ce chapitre, tous les diagrammes de classes sont présentés avec UML [OMG03]). Il définit des super-classes `NodeData` et `ArcData` qui représentent les données portées respectivement par les nœuds et par les arcs du graphe. Dans les deux exemples précédents, cela signifie que des sous-classes `Flow` (pour modéliser les graphes de flot) et `Geographic` (pour modéliser les graphes géographiques) doivent être définies. Un algorithme manipulant, par exemple, des graphes de flot attendra des données sur les arcs qui appartiennent à la classe `Flow`. Ainsi, pour utiliser cet algorithme avec un autre type de graphe, la classe `Flow` doit être spécialisée et quelques-unes de ses méthodes surchargées. De part la nature des algorithmes développés sur de tels graphes, ces méthodes peuvent être appelées très souvent et la liaison dynamique induite peut alors conduire à une certaine inefficacité.

Considérant cette remarque, un modèle utilisant la généricité est proposé à la figure D.2. Les classes `NodeData` et `ArcData` deviennent des *concepts* (ou des *interfaces*) [AUS99], respectivement des paramètres `TN` et `TA` du modèle de classe `Graph`. De cette manière, les types des graphes de flot ou géographiques peuvent être instanciés simplement en définissant les classes `Flow` et `Geographic`, qui doivent satisfaire respectivement les concepts `ArcData` et `NodeData` de sorte que le graphe puisse les manipuler.

L'inconvénient majeur de cette approche est que tous les nœuds (respectivement tous les arcs) doivent porter le même type de données. Il s'agit d'un problème récurrent lorsqu'on doit choisir entre l'héritage et la généricité pour concevoir une structure de données. Cependant, la conception générique peut être combinée à la conception par héritage pour en conserver toute la flexibilité, mais l'efficacité gagnée par la première sera perdue par la liaison dynamique induite dans la seconde.

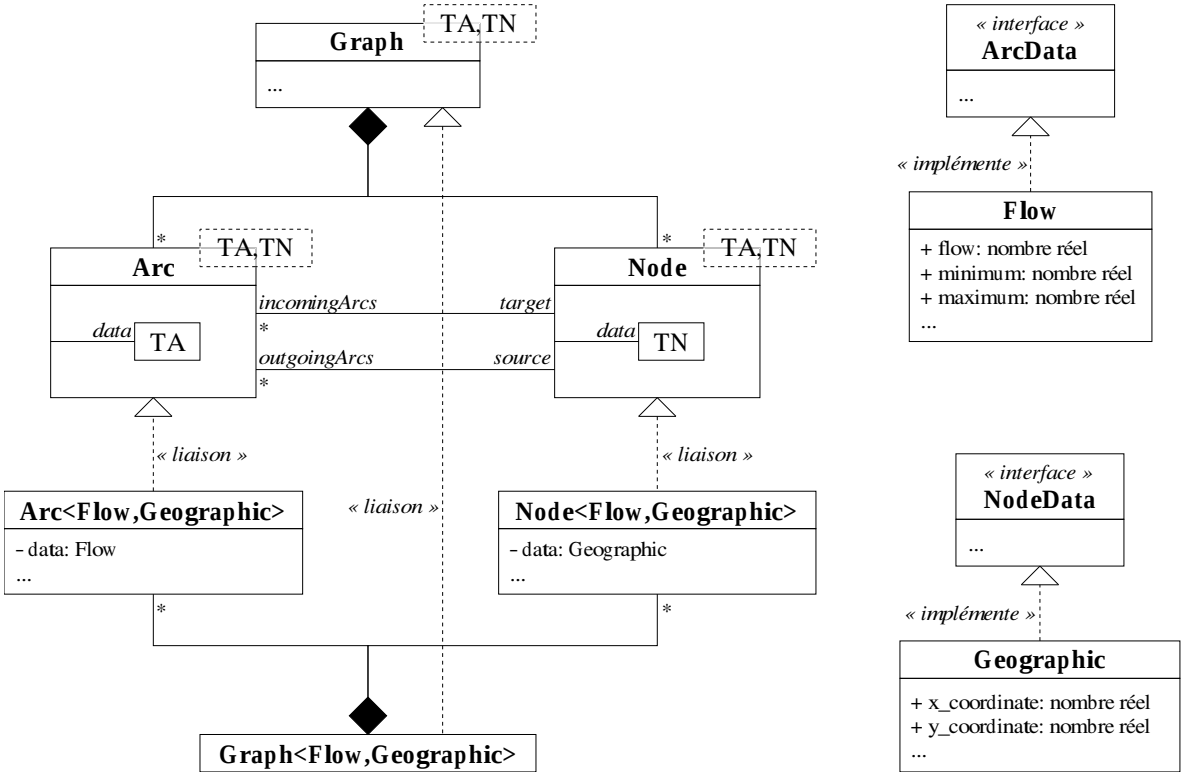


FIG. D.2 – Graphe modélisé par généricité.

Nous proposons de comparer ces deux structures, en mesurant sur une implémentation le temps d'accès à la donnée de flot portée par un arc. Nous comparons trois situations :

1. Avec l'approche par héritage, on tente d'accéder à la donnée de flot. Lorsque la donnée de l'arc est récupérée, elle est de type `ArcData`, il faut alors la convertir en une donnée de type `Flow`. Cette conversion descendante n'est pas garantie, sa validité doit être vérifiée au moment de son exécution, ici avec le mot-clé C++ `dynamic_cast` :

   ```
   dynamic_cast<Flow &>(arc.getData()).getFlow();
   ```

2. Toujours à partir de l'approche par héritage, il est possible d'imposer une conversion sans garantie qui n'effectue aucune vérification dynamiquement, en utilisant le mot-clé C++ `static_cast` :

   ```
   static_cast<Flow &>(arc.getData()).getFlow();
   ```

3. Avec l'approche par généricité, aucune conversion n'est nécessaire, puisque le code est instancié spécifiquement pour des données de type `Flow` sur les arcs :

   ```
   arc.getData().getFlow();
   ```

Les résultats numériques [1] ont été obtenus avec un compilateur GCC 3.2, sous l'environnement Cygwin et sur un processeur Athlon XP 1800+. Le tableau D.1 présente donc une comparaison des deux structures dans les trois situations exposées précédemment. Un graphe de 10000 arcs a été généré et la somme des flots des arcs a été calculée. Le temps mesuré est celui de 10000 répétitions de ce processus.

Approche	Temps d'exécution
(1) Héritage avec conversion dynamique	36.1 s
(2) Héritage avec conversion statique	31.2 s
(3) Généricité	7.4 s

Tab. D.1 – Comparaison d'un graphe par héritage avec un graphe par généricité.

On voit tout de suite le surcoût induit par la liaison dynamique. En ce qui concerne les deux types de conversion, le bénéfice apporté par l'approche statique semble bien faible par rapport aux risques que son utilisation engendre (l'application est moins robuste).

D.2.2. Indépendance des structures de données

La conception de structures de données par une approche générique se révèle être efficace et est largement utilisée (par exemple la STL, *Standard Template Library* [AUS99]). Mais elle n'est pas suffisante pour assurer l'indépendance des algorithmes par rapport aux structures de données qu'ils manipulent. Pour réaliser cet objectif, la solution communément employée consiste à proposer une ou plusieurs classes qui deviennent des interfaces entre l'algorithme et la structure de données qu'il manipule. Généralement, une interface est proposée pour chaque catégorie d'opérations sur la structure de données. Par exemple, pour parcourir une structure de données, l'interface très connue *itérateur* [GHJV95] (également présente dans la STL) est utilisée. Nous pouvons également imaginer une interface pour accéder à des informations globales sur la structure, comme sa taille par exemple.

[1] L'implémentation complète des tests présentés dans ce chapitre est disponible à l'adresse : `http ://www.nawouak.net/ ?cat=informatics.generic_or+lang=fr`

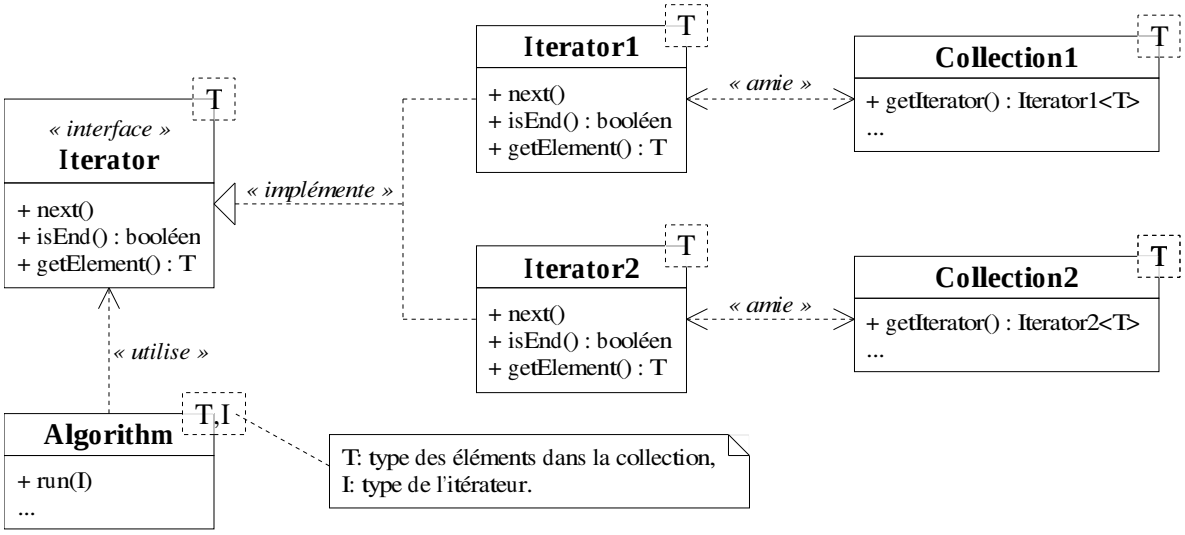


FIG. D.3 – Algorithme paramétré sur le type de l'itérateur.

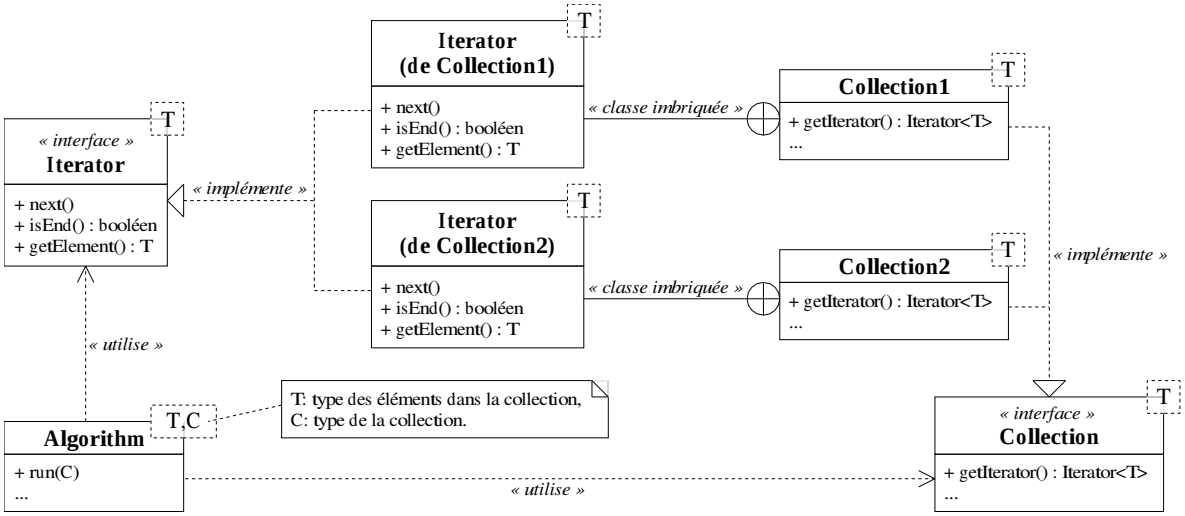


FIG. D.4 – Algorithme paramétré sur le type de la structure de données.

Nous considérons une simple interface itérateur, mais notre discussion peut être généralisée à n'importe quelle autre interface. Certaines conceptions (notamment la STL) proposent que l'algorithme reçoive directement les itérateurs au lieu de la structure de données. De cette manière, l'algorithme est complètement indépendant de la structure de données, comme l'exemple des collections à la figure D.3. Cependant, si plusieurs itérateurs sont requis par l'algorithme, l'utilisateur doit tous les fournir, ce qui conduit à une rupture partielle de l'encapsulation de l'algorithme : l'utilisateur doit connaître certains détails afin de fournir les itérateurs adéquats. L'exemple suivant illustre comment utiliser la collection à partir de la

modélisation de la figure D.3 :

```
void Algorithm<T,I>::run(I i) {
 while (!i.isEnd()) {
  ...i.getElement()...
  ...
  i.next();
 }
}
```

Une meilleure conception serait de proposer une version paramétrable de l'algorithme où le paramètre est le type de la structure de données que l'algorithme manipule (voir la figure D.4). La collection est fournie directement à l'algorithme, alors que le modèle de l'algorithme est indépendant du type de la collection. Cependant, la structure de données doit implémenter un concept spécifique : avec l'exemple de l'itérateur, la collection doit fournir des méthodes qui créent des itérateurs sur sa propre structure. Le type de l'itérateur doit également être fourni par la structure de données comme le montre la figure D.4 avec le type imbriqué `Iterator`. Cela signifie qu'une collection complètement indépendante aura besoin d'un adaptateur pour que l'algorithme puisse l'utiliser. L'exemple suivant illustre comment utiliser la structure de données à partir de la modélisation de la figure D.4 :

```
void Algorithm<C,T>::run(C & c) {
 C::Iterator i = c.getIterator();

 while (!i.isEnd()) {
  ...i.getElement()...
  ...
  i.next();
 }
}
```

Dans notre implémentation, nous avons analysé l'impact d'utiliser des itérateurs pour parcourir les listes d'arcs et de nœuds d'un graphe. Pour cela, nous avons repris la procédure de la section 2.1, et nous avons mesuré, dans les mêmes conditions, son temps d'exécution (test n^{o}4). Le tableau D.2 montre qu'utiliser des itérateurs ne ralentit pratiquement pas la conception, du moment qu'il est possible de dérouler les méthodes de l'itérateur.

Approche	Temps d'exécution
(3) Généricité (sans itérateur)	7.4 s
(4) Généricité (avec itérateur)	7.5 s

Tab. D.2 – Impact de l'utilisation des itérateurs.

D.3. Vers des algorithmes génériques

Un algorithme générique doit être indépendant des composants qu'il manipule, les structures de données comme les algorithmes. La section précédente explique comment abstraire les premières, nous rappelons maintenant une solution de conception classique permettant la même chose pour les seconds. Nous discutons ensuite de diverses techniques pour rendre un algorithme extensible.

D.3.1. Abstraction des algorithmes

Nous avons vu à la section 2 qu'un algorithme peut être modélisé par une classe avec une méthode, `run()` par exemple, qui est appelée pour exécuter l'algorithme. De plus, une telle classe peut posséder des attributs définissant les paramètres de l'algorithme, ainsi les instances représenteront l'algorithme avec différents paramètres. Reposant sur cette modélisation, le patron de conception (ou *design pattern*) *stratégie* [GHJV95] permet de rendre des composants indépendants d'un algorithme : comme il est représenté par une classe, il est possible de définir une super-classe abstraite qui rassemble tous les algorithmes qui résolvent un même problème. Comme le montre la figure D.5, le problème classique du plus court chemin entre deux nœuds dans un graphe (classe abstraite `ShortestPathAlgo`) peut être résolu avec divers algorithmes [AMO93] : `BellmanAlgo`, `DijkstraAlgo`...

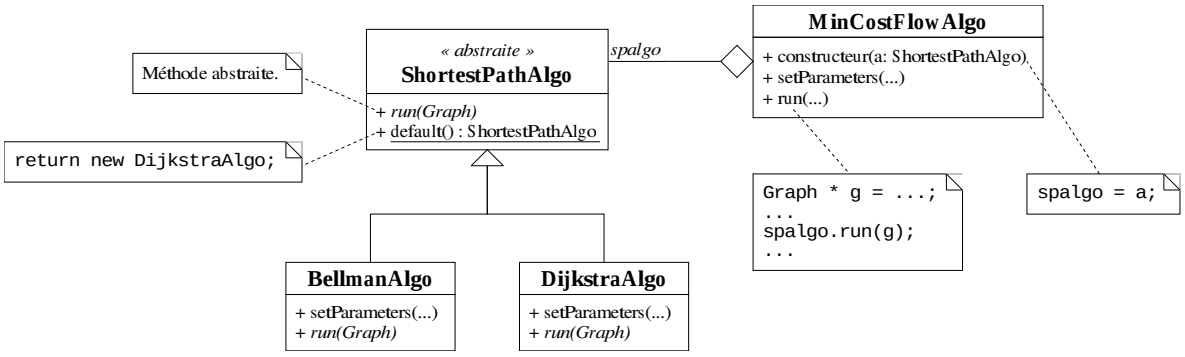


FIG. D.5 – Abstraction des algorithmes.

La méthode `run()` de la classe `ShortestPathAlgo` est abstraite, les sous-classes doivent donc la surcharger. De cette manière, les différents algorithmes de plus court chemin deviennent interchangeables dans n'importe quel algorithme qui manipule un objet `ShortestPathAlgo` (par exemple `MinCostFlowAlgo`). Le polymorphisme de la méthode `run()` n'aura que peu d'impact sur l'efficacité totale de la conception, car les algorithmes sont supposés avoir un comportement complexe et long, donc le temps d'exécution requis dans le mécanisme d'appel à la méthode sera insignifiant comparé au temps d'exécution de la méthode même (et peu de gains sont à envisager en déroulant la méthode).

Deux approches sont envisageables pour paramétrer les algorithmes. La première consiste à proposer une méthode abstraite à arguments variables (`setParameters()` par exemple) dans l'interface de la classe `ShortestPathAlgo`. Toute implémentation est possible : passage par chaînes de caractères, vecteur d'arguments héritant d'une même super-classe... La méthode `setParameters()` analyse alors cette liste d'arguments pour initialiser les paramètres de l'algorithme. Pour des raisons d'efficacité, il est préférable d'éviter les arguments variables et de définir une méthode spécifique `setParameters()` dans chaque algorithme.

```
BellmanAlgo     s;
MinCostFlowAlgo f(s);

s.setParameters(...);
f.setParameters(...);
f.run(...);
```

L'exemple précédent, reposant sur la modélisation de la figure D.5, montre qu'il est possible de choisir quel algorithme de plus court chemin utiliser à l'intérieur de l'algorithme de flot de

coût minimum au moment de l'exécution. L'algorithme doit être créé et paramétré avant qu'il ne soit utilisé dans la méthode `run()` de l'algorithme de flot de coût minimum. Notons que le rôle des méthodes `setParameters()` peut être assuré par les constructeurs des classes. Il est également important de fournir une méthode dans la classe abstraite `ShortestPathAlgo` pour retourner à l'utilisateur final un objet algorithme par défaut, appartenant à l'une de ses classes concrètes. En général, il s'agira de la classe qui est reconnue comme la plus efficace, mais nous pouvons imaginer une approche plus sophistiquée où, par exemple, une analyse de la structure du graphe permet de sélectionner l'algorithme le plus performant ou le plus adapté pour résoudre un problème spécifique sur ce graphe.

D.3.2. Extension des algorithmes

Cette section discute de trois manières de rendre un algorithme extensible, l'idée étant de déléguer certaines parties de son code dans des méthodes séparées qui peuvent être remplacées par l'utilisateur. De cette manière, le comportement global de l'algorithme peut être modifié alors que la plus grande partie de son code n'a pas été (et ne peut pas être) altérée. En outre, l'utilisateur n'a pas besoin de connaître tous les détails concernant l'implémentation de l'algorithme, seules quelques informations pertinentes sur les méthodes qu'il peut remplacer sont nécessaires.

Approche par méthode virtuelle

Le patron de conception *méthode paramètre* (*template method* [GHJV95]) est une solution classique pour rendre un algorithme extensible. Il permet d'externaliser des parties de la méthode `run()` d'un algorithme dans des méthodes virtuelles (par exemple `operation1()` et `operation2()` à la figure D.6) appelées les *méthodes paramètres*. Ainsi, par le mécanisme d'héritage, ces méthodes peuvent être surchargées pour modifier leur comportement, laissant le corps de `run()` inchangé.

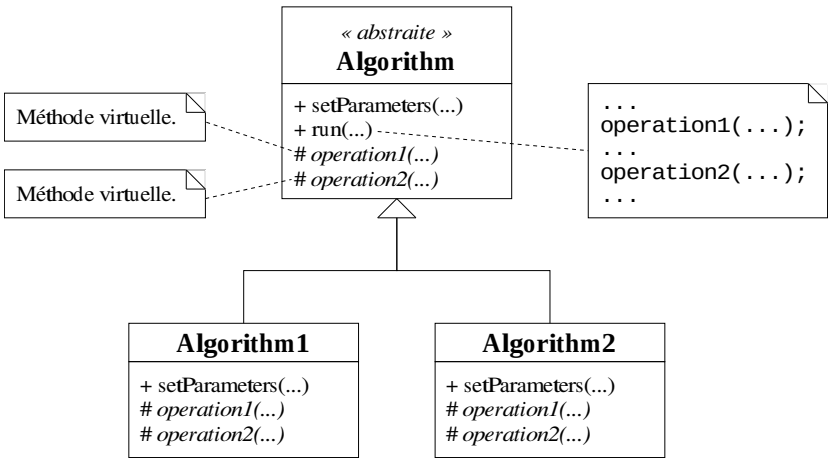


Fig. D.6 – Extension d'un algorithme, approche par méthode virtuelle.

L'inconvénient majeur de cette approche est évidemment l'emploi de la liaison dynamique qui peut conduire à une certaine inefficacité, en particulier quand les méthodes paramètres ont un temps d'exécution bref et qu'elles souvent appelées. Un autre inconvénient est la rigidité

dans l'extension d'un algorithme : il est impossible, au moment de l'exécution, de proposer une extension des méthodes différente de celles définies par les sous-classes de l'algorithme.

Approche par visiteur abstrait

Pour rendre l'extension de l'algorithme plus flexible, la notion de *visiteur* est introduite dans [GHJV95]. Elle propose d'embarquer les méthodes paramètres dans des objets. Plus précisément, un visiteur possède des méthodes qui correspondent aux méthodes paramètres. Pour être opérationnel, l'algorithme doit posséder un attribut représentant le visiteur, qui fournit les parties manquantes de sa méthode `run()`.

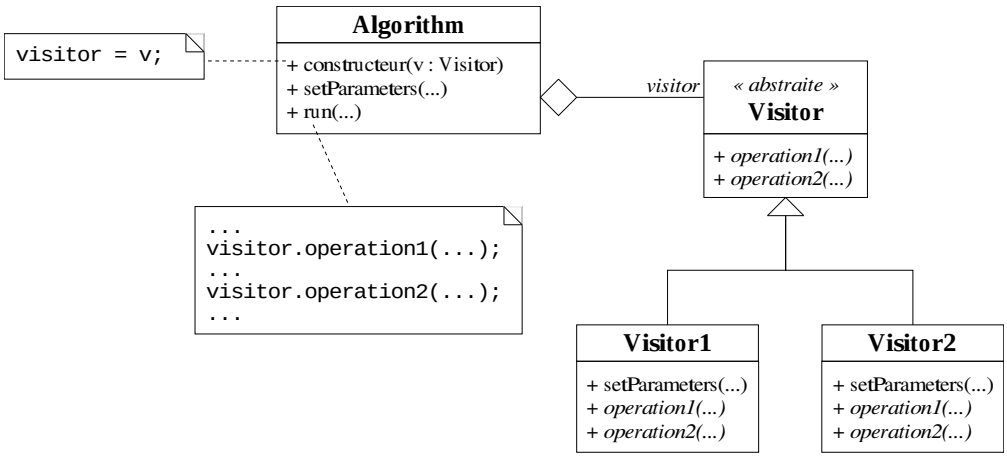


FIG. D.7 – Extension d'un algorithme, approche par visiteur abstrait.

Le visiteur peut être fourni à l'algorithme pendant sa construction, ou plus tard, mais avant l'appel à la méthode `run()`, ou en dernier lieu en tant qu'argument de la méthode `run()`. A la figure D.7, à l'intérieur de la méthode `run()` de la classe `Algorithm`, un objet `visitor` qui implémente l'interface `Visitor` est utilisé pour appeler ses méthodes paramètres embarquées `operation1()` et `operation2()`.

L'exemple suivant montre la flexibilité de cette approche. Il est possible de décider, pendant l'exécution, quel visiteur utiliser pour exécuter l'algorithme. Cependant, l'inconvénient majeur, lié au polymorphisme des méthodes paramètres, demeure.

```
Visitor1  v;
Algorithm a(v);

v.setParameters(...);
a.setParameters(...);
a.run(...);
```

Approche par interface de visiteur

Pour finalement éviter la liaison dynamique, le visiteur doit être un paramètre, non pas de la méthode `run()`, mais de la classe `Algorithm` elle-même. Ce qui signifie que la classe devient paramétrable avec le type du visiteur en paramètre. Ainsi, comme le montre la figure D.8, l'algorithme possède un attribut représentant un visiteur qui doit satisfaire un concept `Visitor`.

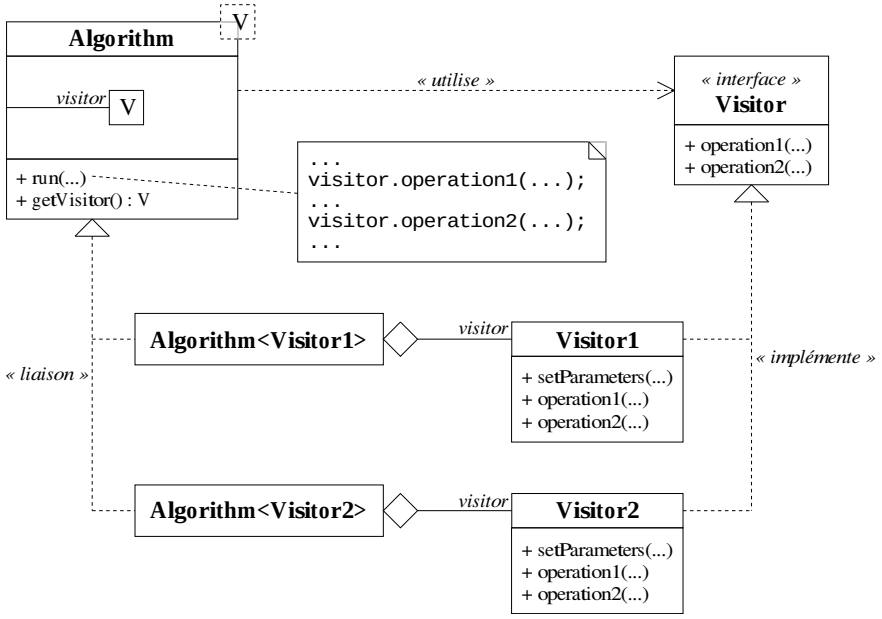


FIG. D.8 – Extension d'un algorithme, approche par interface de visiteur.

L'exemple suivant illustre cette modélisation. Cette approche est très similaire à la précédente, mais la liaison dynamique est évitée, ainsi il n'y a plus de perte d'efficacité. Néanmoins, la flexibilité proposée avec l'approche par visiteur abstrait est perdue. Comme dans l'approche précédente, le visiteur peut être fourni directement en tant qu'argument au constructeur de `Algorithm`, au lieu d'être automatiquement créé par l'algorithme. De cette manière, l'interface de visiteur peut être combinée avec l'approche par visiteur abstrait pour fournir une certaine flexibilité.

```
Algorithm<Visitor1> a;

a.getVisitor().setParameters(...);
a.setParameters(...);
a.run();
```

Comparaison des approches

Pour conclure, nous proposons une comparaison pratique des approches présentées dans ce chapitre pour étendre un algorithme. Nous avons choisi d'implémenter un algorithme de plus court chemin possédant un visiteur dont le rôle est de fournir la longueur d'un arc. Ainsi, si l'on considère une classe `Route` avec deux attributs `distance` et `speed`, il est possible, en proposant deux visiteurs différents (ou deux sous-classes différentes pour la première approche), de calculer un plus court chemin aussi bien en terme de temps qu'en terme de distance, sans modifier une seule ligne de l'algorithme.

```
class TimeVisitor {
 public : int getLength(Route & r) { return r.distance*r.speed; }
};

class DistanceVisitor {
 public : int getLength(Route & r) { return r.distance; }
};
```

Sur un graphe de 100000 arcs, nous proposons de rechercher, entre deux points tirés au hasard, le plus court chemin, tout d'abord en terme de temps, puis en terme de distance, par deux appels successifs à l'algorithme (soit avec des visiteurs différents, soit avec des versions différentes de l'algorithme, selon l'approche). Le temps mesuré est celui de 1000 répétitions de ce processus.

Approche	Temps d'exécution
(a) Méthode virtuelle	20.9 s
(b) Visiteur abstrait	20.9 s
(c) Interface de visiteur	18.7 s
(d) Classique	18.7 s

Tab. D.3 – Comparaison des approches pour étendre un algorithme.

Nous comparons les différentes approches à la solution classique qui consiste à écrire deux versions distinctes de l'algorithme : l'une dédiée au temps, l'autre à la distance. Les résultats proposés dans le tableau D.3 montrent que les écarts entre les différentes approches ne sont pas tellement importants. Il est tout à fait normal que les méthodes *(a)* et *(b)* soient équivalentes, le coût d'appel à la méthode `getLength()` est le même puisqu'aucun déroulement n'est possible. En revanche, pour la méthode *(c)*, ce déroulement est possible, ce qui explique des performances équivalente à l'approche classique *(d)*. Le peu d'écart entre les méthodes, seulement deux secondes, s'explique ici par le fait que l'appel à la méthode `getLength()` n'est pas très fréquent par rapport à la masse des opérations de l'algorithme. Un test similaire à celui présenté à la section 2 qui consisterait à simplement faire la somme des longueurs sur les arcs (toujours avec des visiteurs) creuserait les écarts entre les méthodes.

D.3.3. Obtenir une bonne généricité

Finalement, pour obtenir une "bonne" généricité des algorithmes, il semble important d'appliquer le patron de conception stratégie avec les solutions proposées ici pour rendre les composants indépendants, et de le combiner soit avec l'approche par interface de visiteur (quand l'efficacité est importante), soit avec l'approche par visiteur abstrait (quand la flexibilité est préférée), afin de permettre une extensibilité suffisante pour l'algorithme. Notons que la STL propose la notion de *foncteur* qui est très similaire à la notion d'interface de visiteur. D'autres approches sont proposées, par exemple dans [DGD01] qui définit des versions génériques de plusieurs patrons de comportement introduits dans [GHJV95].

D.4. Gestion d'extensions pour les structures de données

Lors de la conception d'algorithmes génériques, il est souvent nécessaire d'étendre une structure de données, de manière à ce que ses éléments fournissent des attributs additionnels que l'algorithme puisse utiliser temporairement. Par exemple, pour résoudre un problème de flot de coût minimum certains algorithmes requièrent l'affectation d'un potentiel aux nœuds du graphe. Cependant, les nœuds des graphes de flot ne possèdent pas de telles données, et on ne peut pas attendre de la part de l'utilisateur de l'algorithme qu'il ajoute ces données, car cela briserait l'encapsulation du composant.

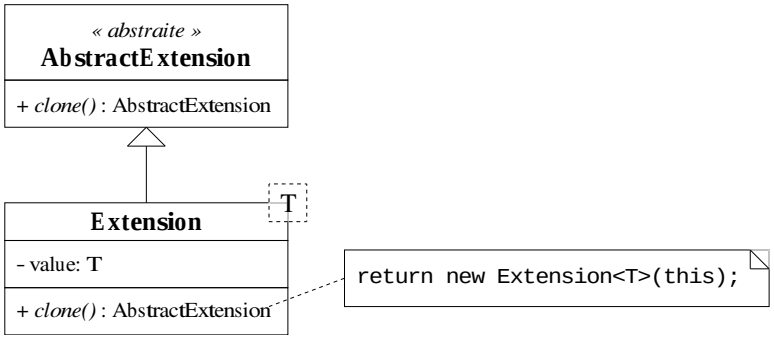


FIG. D.9 – Classe `Extension`.

Pour répondre à ce problème, la bibliothèque BGL (*Boost Graph Library* [SLL02]) propose le concept de *property map*. Même si elle fournit un cadre de travail totalement générique pour accéder aux données additionnelles, elle n'explique pas spécifiquement comment ajouter ces données en premier lieu. Une solution pourrait être de stocker les données dans une structure indépendante du graphe. L'algorithme peut alors associer des données à un nœud à travers cette structure de données associative. Cependant, une telle solution ne peut pas supporter à la fois la modification du graphe et un accès aux données additionnelles à partir d'un nœud en temps constant ($O(1)$ opérations). Un compromis entre ces deux propriétés doit être fait : par exemple un tableau de données où chaque nœud a un indice arbitraire (mais supprimer ou ajouter un nœud implique des modifications à la fois dans l'indexation et dans le tableau) ; ou un arbre binaire qui associe des données à chaque nœud (mais accéder, supprimer ou ajouter un nœud nécessite $O(\log n)$ opérations).

Dans cette section, nous nous intéressons à concevoir une solution qui propose un accès direct (i.e. en temps constant) aux données additionnelles à partir de l'objet nœud même. La première idée est d'ajouter un attribut "libre" aux nœuds du graphe. Cet attribut est une référence sur un objet d'une classe `AbstractExtension`. Quand un graphe est construit, aucune donnée n'est pointée. Ensuite, si un algorithme a besoin d'ajouter des données, il peut faire pointer l'attribut "libre" sur un objet appartenant à une classe dérivant de `AbstractExtension`. La classe paramétrable `Extension` (voir la figure D.9) est proposée pour offrir une manière générique d'encapsuler une entité à l'intérieur d'un objet avec l'interface `AbstractExtension`.

Cette première approche nécessite qu'un algorithme n'utilise un attribut "libre" qu'après avoir mémorisé les données qu'un autre algorithme aurait pu y stocker, et qu'il restaure ce dernier une fois son exécution terminée. Il s'agit typiquement d'une pile, la seconde idée est donc de remplacer l'attribut "libre" par une pile d'attributs "libres". Quand un algorithme a besoin d'ajouter des données, il doit les placer au sommet de la pile et les retirer après son exécution.

Cependant, l'exécution de certains algorithmes peut être itérative, et entre deux itérations, d'autres actions peuvent être exécutées. Par exemple, dans l'apprentissage avec élagage d'un réseaux de neurones, il y a deux algorithmes itératifs indépendants : l'algorithme d'apprentissage qui modifie, à chaque itération, les poids du réseaux de neurones, et l'algorithme d'élagage qui peut retirer, à chaque itération, des arcs qui se révèlent inutiles. Le processus complet consiste à exécuter quelques itérations de l'apprentissage, ensuite une itération d'élagage, et à répéter ceci jusqu'à ce que certaines conditions soient satisfaites. A la fois l'apprentissage et l'élagage nécessitent l'ajout de données additionnelles sur les nœuds du graphe

qui doivent rester entre deux itérations de chaque algorithme. Cela signifie que l'ordre dans lequel les algorithmes ajoutent les données aux nœuds ne peut pas être modélisé par une pile. N'importe quel algorithme peut ajouter ou retirer, à tout moment, ses propres données des nœuds.

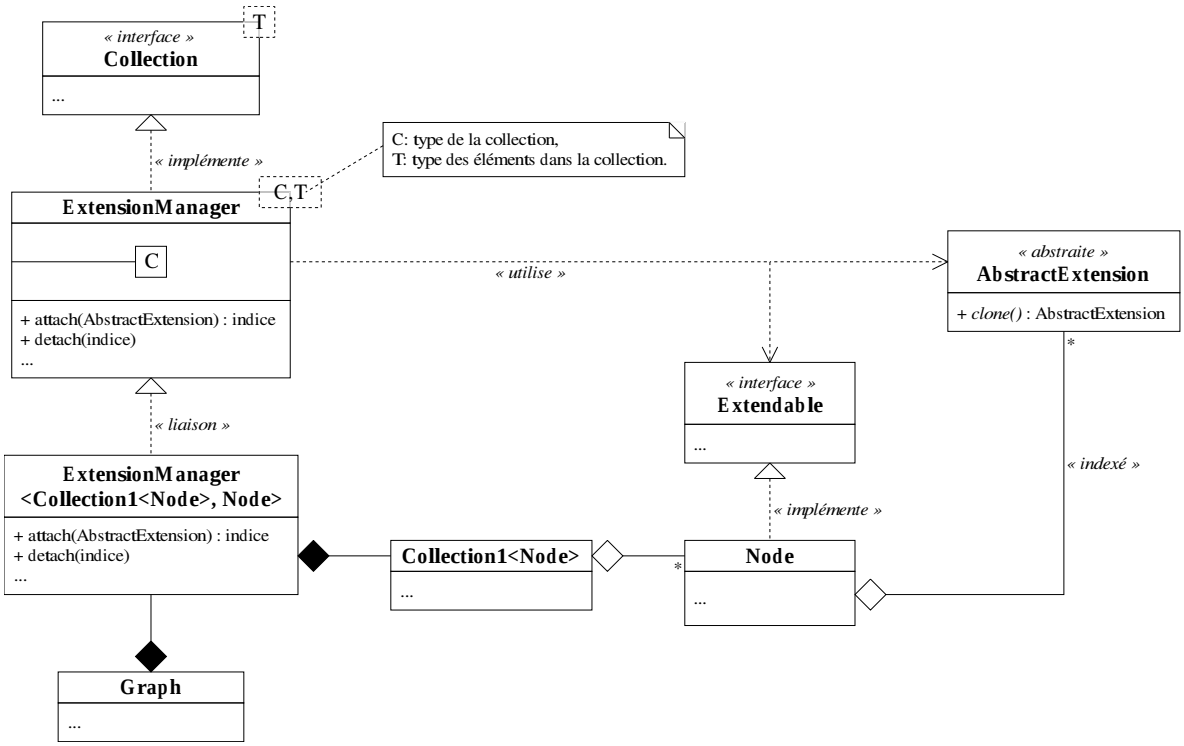


FIG. D.10 – Modélisation de la gestion de données additionnelles.

La figure D.10 présente une solution pour concevoir une structure de données qui peut gérer l'insertion ou la suppression de données dans un ensemble d'éléments. L'exemple d'une collection de nœuds dans un graphe est considérée. Au lieu de manipuler une collection de nœuds directement, avec la classe `Collection1<Node>` par exemple (voir la figure D.4), le graphe manipule un objet de la classe `ExtensionManager` qui implémente l'interface `Collection`, ainsi aucun changement dans le code de la classe `Graph` n'est requis (excepté pour la déclaration de la collection de nœuds). Cette classe `ExtensionManager` est un adaptateur encapsulant une collection ; tout appel aux méthodes de l'interface `Collection` est délégué à la collection interne.

Le rôle d'un objet `ExtensionManager` est de gérer l'ajout ou la suppression d'un objet `Extension` de chaque nœud de l'ensemble qu'il encapsule. Ainsi, un algorithme qui désire ajouter une extension appelle sa méthode `attach()` avec un modèle de l'extension à cloner et à placer sur chaque nœud. Un indice est retourné indiquant la position de l'extension dans les listes indexées que les nœuds possèdent. Ainsi, cela permet à l'algorithme de demander directement à un nœud une extension spécifique, en utilisant l'indice. Finalement, pour supprimer une extension, l'algorithme appelle la méthode `detach()` avec l'indice en argument, de manière à ce que le gestionnaire `ExtensionManager` sache quelle extension retirer des nœuds. Pour manipuler les ensembles d'extensions, le gestionnaire utilise l'interface `Extendable` implémentée par les nœuds. Pour être plus générique, l'indice utilisé par les méthodes `attach()`

et `detach()` peut être remplacé par un objet (notamment le concept de *property map* proposé dans BGL [SLL02]) dont le rôle est de fournir l'extension spécifique de chaque nœud, cachant ainsi la manière dont les extensions sont stockées en mémoire.

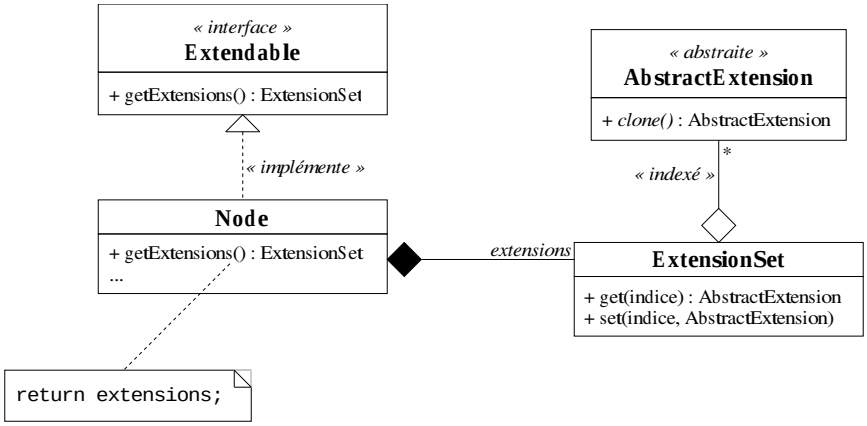


FIG. D.11 – Interface `Extendable` implémentée avec la délégation.

Deux solutions sont envisageables pour implémenter l'interface `Extendable`. Premièrement, la classe `Node` peut déléguer la gestion de la liste d'extensions à une autre classe (c'est le rôle de `ExtensionSet` à la figure D.11). Deuxièmement, l'interface `Extendable` peut devenir une classe qui gère la liste d'extensions (comme `ExtensionSet` dans la conception précédente), et la classe `Node` hérite de la classe `Extendable` (voir la figure D.12). Cette spécialisation est efficace car aucune liaison dynamique n'est impliquée. Dans la seconde conception, la partie implémentation de la classe est héritée de la classe `Extendable`. Cela soulève des problèmes avec les langages qui n'autorisent pas l'héritage multiple de la partie implémentation, ce qui signifie que `Node` ne peut pas hériter de l'implémentation d'une autre classe, or cela pourrait être nécessaire dans certaines conceptions.

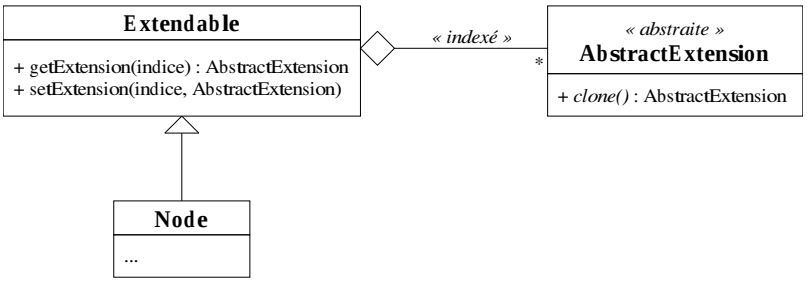


FIG. D.12 – Interface `Extendable` implémentée avec la spécialisation.

En termes de maintenance et de réutilisabilité, cette conception autorise l'ajout d'un objet `ExtensionManager` sans modifier la structure de données qui possède la collection originale, `Graph` par exemple. La structure de la classe qui agrège les extensions n'a pas à être modifiée, elle doit juste posséder un attribut de la classe `ExtensionSet` ou dériver de la classe `Extendable`, les deux solutions fournissant un ensemble d'extensions. La liaison dynamique a été évitée autant que possible, cependant les algorithmes doivent convertir les extensions de `AbstractExtension` en `Extension<T>` de manière descendante. Cela signifie qu'une vérifica-

tion de type au moment de l'exécution est requise, conduisant généralement à une certaine inefficacité, car les extensions sont des données simples pour lesquelles nous pouvons supposer un accès très fréquent. L'efficacité est garantie seulement si la vérification de type peut être évitée, ce qui est possible, comme nous l'avons vu à la section 2, avec des langages comme C++.

Dans notre implémentation de l'algorithme de plus court chemin utilisée pour des comparaisons pratiques, deux données peuvent être gérées comme des extensions : pour les besoins de l'algorithme, un potentiel et un marquage doivent être associés à chaque nœud du graphe. Dans les tests de la section 3, nous avons placé ces données en tant qu'attributs de la classe `Node`, ce qui est la solution la plus efficace, mais elle brise totalement l'encapsulation de l'algorithme. Nous proposons maintenant de reprendre des tests pratiques présentés depuis le début ddu chapitre : *(3)* et *(4)* de la section 2, *(a)* à *(d)* de la section 3. Mais cette fois-ci, avec des classes `Node` et `Arc` qui implémentent l'interface `Extendable`, et une classe `Graph` qui, au lieu de posséder des vecteurs de nœuds et d'arcs, possède des `ExtensionManager` pour stocker les nœuds et les arcs.

Test	Sans extension	Avec extension
(3) Généricité (sans itérateur) (cf. section 2)	7.4 s	7.4 s
(4) Généricité (avec itérateur) (cf. section 2)	7.5 s	7.5 s
(a) Méthode virtuelle (cf. section 3)	20.9 s	21.5 s
(b) Visiteur abstrait (cf. section 3)	20.9 s	21.5 s
(c) Interface de visiteur (cf. section 3)	18.7 s	19.2 s
(d) Classique (cf. section 3)	18.7 s	19.2 s
(e) Interface de visiteur + ext. dynamique		24.8 s
(f) Interface de visiteur + ext. statique		19.8 s

Tab. D.4 – Impact du mécanisme d'extension.

Dans les tests *(a)* à *(d)*, les extensions sont présentes mais ne sont pas exploitées par l'algorithme de plus court chemin. Nous proposons donc les tests *(e)* et *(f)* où l'algorithme utilise les extensions, il n'y donc plus de brèche dans l'encapsulation. Le test *(e)* propose une conversion avec vérification dynamique pour accéder à l'extension, alors que le test *(f)* propose une conversion statique (qui n'affaiblit pas vraiment la robustesse de l'application puisque les extensions sont entièrement contrôlées par l'algorithme, et que l'utilisateur ne doit normalement pas intervenir sur ces données). Excepté dans les tests *(3)* et *(4)*, on s'aperçoit que l'installation du mécanisme d'extension engendre un léger surcoût, dû principalement au fait que les nœuds et les arcs sont d'une taille un peu plus importante. Il faut noter également que la conversion dynamique utilisée dans le test *(e)* induit un surcoût important par rapport au test *(f)*. Il est donc recommandé, si le langage le permet, d'employer la conversion statique. En résumé, le mécanisme d'extension entraîne, pour nos tests, un surcoût de moins de 3 % lorsqu'il n'est pas utilisé, et de l'ordre de 6 % lorsqu'un algorithme le sollicite, ce qui peut être acceptable selon l'application.

D.5. Maintenir plusieurs modèles d'un même problème

Une difficulté récurrente est de manipuler plusieurs modèles d'un problème en même temps. Par exemple, un graphe peut être représenté sous la forme d'une matrice. Ainsi,

un problème d'optimisation peut être modélisé sur la base d'une structure de graphe (par exemple, la classe `Graph` paramétrée de façon à représenter le problème), ou bien sous la forme d'un programme linéaire, une forme matricielle (représentée par la classe `Matrix`). Un algorithme peut nécessiter de manipuler ces deux formes d'un problème en même temps : effectuer une partie du traitement sur l'objet `Graph`, convertir alors le problème en un objet `Matrix`, réaliser une autre partie du traitement, puis revenir à la représentation sous forme de `Graph` pour intégrer les derniers résultats. Les données de l'objet `Matrix` devront être interprétés pour modifier l'objet `Graph`.

Une première solution nécessite un objet convertisseur qui fournit une méthode de transformation d'un graphe en une matrice, et une autre qui interprète les résultats de la matrice pour les répercuter sur le graphe. L'inconvénient de cette approche est évident : chaque fois qu'une modification est faite sur le graphe, l'objet convertisseur doit être appelé pour reconstruire complètement la matrice.

Une solution plus élaborée peut être proposée : l'un des deux composants, `Graph` ou `Matrix` peut être "virtuel". Cela signifie que seule l'une des deux structures de données existe physiquement, et l'autre est simplement un adaptateur de la première. La figure D.13(a) propose une classe `VirtualMatrix` qui implémente l'interface `Matrix` et encapsule le graphe à convertir. Chaque fois qu'une méthode de l'interface `Matrix` est appelée, l'objet `VirtualMatrix` délègue l'exécution au graphe associé.

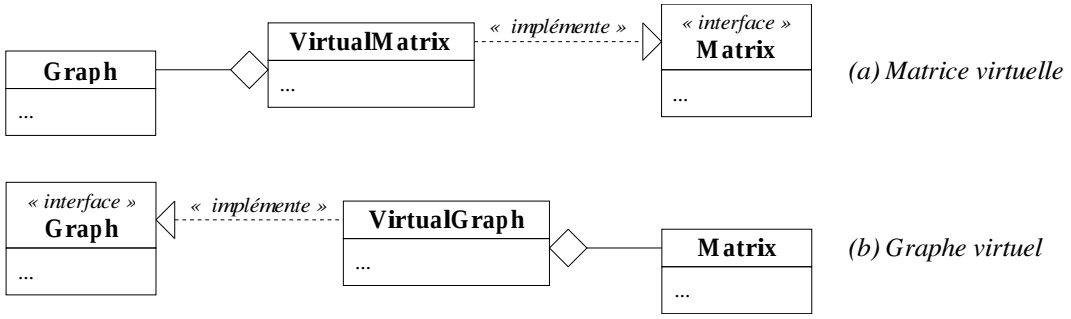


Fig. D.13 – Structure de données virtuelle.

A l'opposé, la figure D.13(b) propose de rendre la structure de graphe virtuelle. Avec cette solution de conception, le graphe (respectivement la matrice) peut être modifié de manière fiable à tout moment, car chaque fois que des informations de la matrice (respectivement du graphe) sont requises, elles sont construites à partir de la structure du graphe (respectivement de la matrice). Cependant, si les méthodes du composant virtuel nécessitent un certain temps pour être exécutées, la conception devient inefficace. Elle doit plutôt être utilisée lorsque les méthodes nécessitent peu de temps d'exécution, le meilleur des cas étant qu'elles ne traduisent qu'une relation triviale entre les éléments de la matrice (respectivement du graphe) et les éléments du graphe (respectivement de la matrice).

La troisième solution consiste à maintenir plusieurs modèles physiques d'un problème en même temps. On dispose alors de deux classes `Graph` et `Matrix`, et lorsqu'une modification survient dans l'objet `Graph`, elle doit être aussitôt répercutée dans l'objet `Matrix` (dans un souci de simplicité, le cas opposé n'est pas considéré). Cela signifie que les algorithmes manipulent un adaptateur de la classe `Graph`, `ObservedGraph` par exemple, qui suit la même interface `GraphInterface`. De plus, le patron de conception *observateur* [GHJV95] est implémenté sur

le graphe observé, ce qui signifie que des objets externes, les observateurs, peuvent demander à leur gestionnaire dans le graphe observé (`ObserverManager`) d'être informés quand certaines opérations surviennent. Le graphe observé décide de notifier ses changements pertinents à un ou plusieurs gestionnaires, qui informent à leur tour les observateurs. Ces derniers, recevant une notification, peuvent choisir ou non de modifier la matrice afin de garder la cohérence avec le graphe. Ainsi, un algorithme peut manipuler `ObservedGraph` comme n'importe quel graphe, et chaque fois qu'une opération pertinente survient, la matrice est modifiée. Cette solution n'est efficace que si peu d'appels aux observateurs sont effectués.

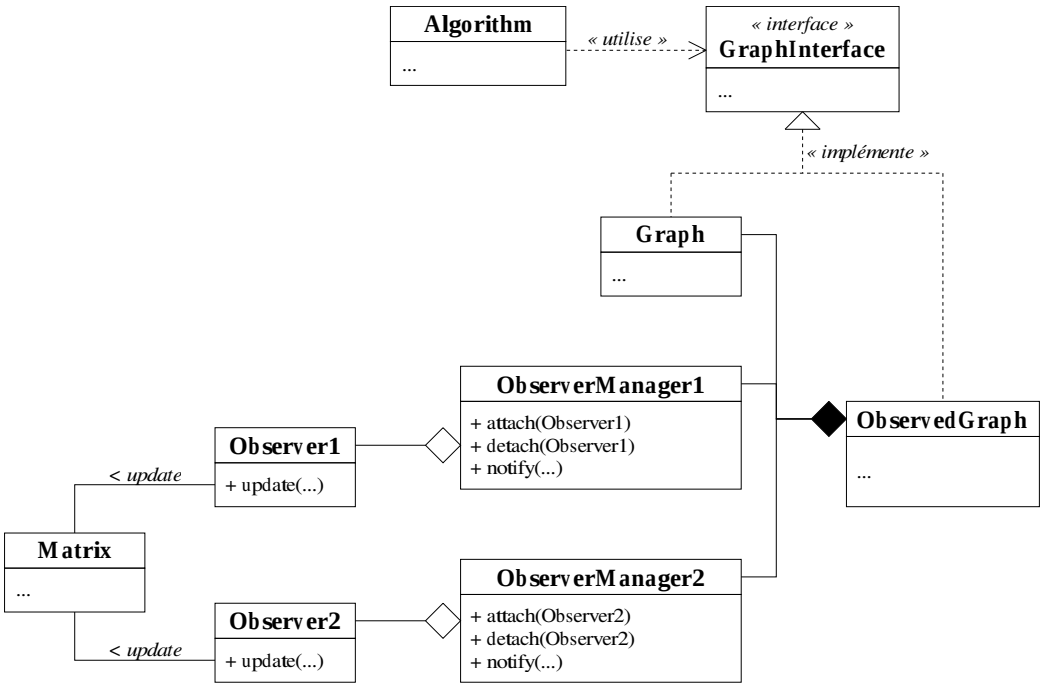


FIG. D.14 – Maintenir plusieurs modèles.

En conclusion, si les modèles ne nécessitent pas d'être maintenus ensemble, la première solution peut être mise en œuvre. S'il est possible d'établir une relation triviale entre les éléments des deux modèles, la seconde solution avec un simple adaptateur est efficace. Finalement, si la relation entre les modèles est plus complexe et que deux modèles physiques sont nécessaires, la troisième approche devra être choisie. Dans toutes ces solutions, la liaison dynamique doit être évitée.

D.6. Conclusion

L'objectif de ce chapitre est de montrer comment il est possible de concevoir des algorithmes génériques tout en maintenant leur efficacité proche de celle d'un développement dédié. Pour rendre un composant générique, deux objectifs doivent être réunis : le composant doit être interchangeable (les composants qui l'utilisent doivent en être le plus indépendant possible) et extensible.

Dans un premier temps, nous avons étudié les techniques couramment employées pour les structures de données. Les solutions sont maîtrisées : les classes paramétrables permettent

l'extensibilité et les itérateurs l'indépendance. Des mesures sur un cas concret montrent qu'aucune perte d'efficacité n'est liée à leur utilisation.

Concernant les algorithmes, leur indépendance est rendue possible grâce au patron de conception stratégie, qui n'engendre qu'un surcoût négligeable. Nous avons proposé trois solutions pour assurer leur extensibilité. Une comparaison pratique montre que l'approche par interface de visiteur, exploitant la généricité, est la meilleure alternative et n'a aucun impact sur l'efficacité de l'algorithme.

A partir de ces constats, nous avons présenté des solutions à des problèmes récurrents en recherche opérationnelle, comme la manipulation d'informations additionnelles sur des structures de données, et le maintien de plusieurs modèles pour un même problème. La technique que nous proposons pour le premier problème évite de rompre l'encapsulation d'algorithmes, tout en n'altérant que légèrement leur efficacité. Pour le second problème, plusieurs solutions sont proposées en fonction de l'interaction souhaitée entre les modèles.

De ces expériences, il ressort que le langage C++, bien que satisfaisant en terme d'efficacité, souffre de quelques lacunes au niveau de la programmation générique. La syntaxe proposée par le langage, ainsi que les techniques mises en œuvre, rendent difficile l'écriture d'algorithmes génériques pour des personnes non initiées. Il serait très intéressant de proposer un préprocesseur au langage qui permettrait de définir un langage plus adapté, comme c'est le cas du langage CLAIRE [CJL02] par exemple. L'absence de contraintes sur les paramètres des classes génériques (par l'intermédiaire des *concepts* notamment) rend également difficile la phase de développement. Les tentatives d'introduction de concepts dans le langage C++ ([MS00] et [SL00]) ne permettent pas un contrôle aussi avancé que les solutions proposées dans d'autres langages (notamment Java [BOW98]).

Tous ces projets ont démarré dans les années 1999 et 2000. Pendant cette même période, des projets de bibliothèques pour les graphes comme BGL (*Boost Graph Library* [SL00]) et GTL (*Graph Template Library* [FPR99]) sont apparus. L'actuel intérêt pour ces bibliothèques montre que les praticiens de recherche opérationnelle commencent à être sensibles aux arguments des programmations objet et générique. Pour dépasser cette première étape, il semble nécessaire de progresser vers un contexte plus formel avec des patrons et une méthodologie pour guider la conception de tels logiciels.

Index

Bibliographie

[AMO93] Ravindra K. AHUJA, Thomas L. MAGNANTI et James B. ORLIN. *Network Flows - Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[ASS78] Arjang A. ASSAD. Multicommodity Network Flows — A Survey. *Networks*, 8 :37–91, 1978.

[AUS99] Matthew H. AUSTERN. *Generic Programming and the STL : Using and Extending the C++ Standard Template Library*. Addison-Wesley, 1999.

[AZA92] Farhad AZADIVAR. A Tutorial on Simulation Optimization. In *Proceedings of the Winter Simulation Conference*, pages 198–204, 1992.

[BAC03] Bruno BACHELET. *Modélisation et Optimisation de problèmes de synchronisation dans les documents hypermédia*. Thèse de Doctorat, LIMOS, Université Blaise Pascal, Clermont-Ferrand, 2003.

[BB01] Ursula BORYCZKA et Mariusz BORYCZKA. Multi-Cast Ant Colony System for Bus Routing Problem. In *4th Metaheuristics International Conference*, 2001.

[BBEN97] Osman BALCI, Anders I. BERTELRUD, Chuck M. ESTERBROOK et Richard E. NANCE. The Visual Simulation Environment. In *11th European Simulation Multiconference*, pages 61–68, 1997.

[BBPS97] Jeffrey BRACA, Julien BRAMEL, Bruce POSNER et David SIMCHI-LEVI. A Computerized Approach to the New-York City School Bus Routing Problem. *IIE Transactions*, 29 :693–702, 1997.

[BCC02] Maurizio BIELLI, Massimiliano CARAMIA et Pasquale CAROTENUTO. Genetic algorithms in bus network optimization. *Transportation Research Part C*, 10 :19–34, 2002.

[BCGT98] Daniel BIENSTOCK, Sunil CHOPRA, Oktay GÜNLÜK et Chih-Yang TSAI. Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming B*, 81 :177–199, 1998.

[BEN62] Jacques F. BENDERS. Partionning procedures for solving mixed variable programming problems. *Numerische Mathematik*, 4 :238–252, 1962.

[BER89] Dimitri P. BERTSEKAS. *Linear Network Optimization*. MIT Press, 1989.

[BFG98] Abdelhamid BENCHAKROUN, Jacques A. FERLAND et Viviane GASCON. "benders decomposition for network design problem with underlying tree structure". *Investigaciòn Operativa*, 6 :165–180, 1998.

[BGAB83] Lawrence D. BODIN, Bruce L. GOLDEN, Arjang A. ASSAD et Michael O. BALL. Routing and Scheduling of Vehicles and Crews. The State of Art. *Computers & Operations Research*, 10(2) :63–211, 1983.

[BH00] Ulrich BLASUM et Winfried HOCHSTÄTTLER. Application of the Branch and Cut Method to the Vehicle Routing Problem. Technical Report ZAIK-2000-386, Zentrum für Angewandte Informatik, 2000.

[BHC94] Robert BOWERMAN, Brent HALL et Paul CALAMAI. The Spacefilling Curve with Optimal Partitioning Heuristic for the Vehicle Routing Problem. *European Journal of Operational Research*, 76 :128–142, 1994.

[BHC95] Robert BOWERMAN, Brent HALL et Paul CALAMAI. A Multiobjective Optimization Approach to Urban School Bus Routing : Formulation and Solution Method. *Transportation Research part A*, 29 :107–123, 1995.

[BJN+98] Cynthia BARNHART, Ellis L. JOHNSON, George L. NEMHAUSER, Martin W.P. SAVELSBERGH et Pamela H. VANCE. Branch-And-Price : Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(2) :316 – 329, 1998.

[BKS02] Georg BAIER, Ekkehard KÖHLER et Martin SKUTELLA. On the k-Splittable Flow Problem. In *Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes In Computer Science*, pages 101–113, 2002.

[BLS03] Luce BROTCORNE, Gilbert LAPORTE et Frédéric SEMET. Ambulance location and relocation problem. *European Journal of Operational Research*, 147 :451–463, 2003.

[BMM94] Anantaram BALAKRISHNAN, Thomas L. MAGNANTI et Prakash MIRCHANDANI. A Dual-based Algorithm for Multi-level Network Design. *Management science*, 40(5) :567–591, 1994.

[BMQ03] Fatihah BENDALI, Jean MAILFERT et Alain QUILLIOT. Méthodes de décomposition et d'agrégation pour le traitement de problèmes de multiflots. *Pesquisa Operacional*, 23(3) :475–498, 2003.

[BON63] O.N. BONDAREVA. Some applications of linear programming methods to the theory of cooperative games. *Problemy Kibernetiki*, pages 119–139, 1963.

[BOR98] Ralf BORNDÖRFER. *Aspects of Set Packing, Partioning and Covering*. PhD Thesis, Technische Universität Berlin, 1998.

[BOW98] Kim B. BRUCE, Martin ODERSKY et Philip WADLER. A Statically Safe Alternative to Virtual Types. In *ECOOP'98 - Lecture Notes in Computer Science*, volume 1445, pages 523–549. Springer-Verlag, 1998.

[BOY97] Florence BOYER. *Conception et routage dans les réseaux de télécommunications : application de la méthode de Benders*. Thèse de Doctorat, LIMOS, Université Blaise Pascal, Clermont-Ferrand, Décembre 1997.

[BQ64] Michel L. BALINSKI et Richard E. QUANDT. On an integer program for a delivery problem. Operations Research. *Operations Research*, 12 :300–304, 1964.

[BRT03] Luciana BURIOL, Mauricio G.C. RESENDE et Mikkel THORUP. Speeding up dynamic shortest path path algorithms. Technical Report TD-5RJ8B, AT&T Labs Research, 2003.

[BS96] Julien BRAMEL et David SIMCHI-LEVI. Probalistic Analysis and Practical Algorithms for the Vehicle Routing Problem with Time Windows. *Operations Research*, 44 :501–509, 1996.

[BS97] Julien BRAMEL et David SIMCHI-LEVI. On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows. *Operations Research*, 45 :295–301, 1997.

[BT88] Dimitri P. BERTSEKAS et Paul TSENG. The relax codes for linear minimum cost network flow problems. FORTRAN codes for Network Optimization. *Annals of Operations Research*, 13 :125–190, 1988.

[BT97] Dimitris BERTSIMAS et John N. TSITSIKLIS. *Introduction to Linear Optimization*. Optimization and Computation Series. Athena Scientific, Belmont, Massachusetts, 1997.

[CBL01] Nitin CHANDRACHOODAN, Shuvra S. BHATTACHARRYYA et K. J. R. LIU. Adaptive Negative Cycle Detection in Dynamic Graphs. In *Proceedings of International Symposium on Circuits and Systems (ISCAS 2001)*, volume V, pages 163–166, Sydney, Australia, May 2001.

[CC02] Zbigniew J. CZECH et Piotr CZARNAS. Parallel simulated annealing for the vehicle routing problem with time windows. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 376–383, Canary Islands, Spain, 2002.

[CCC99] João COUTINHO-RODRIGUES, João Carlos Namorado CLIÍMACO et John R. CURRENT. An interactive bi-objective shortest path approach : searching for unsupported nondominated solutionss. *Computers & Operations Research*, 26 :789–798, 1999.

[CDD+99] Jean-François CORDEAU, Guy DELSAULNIERS, Jacques DESROSIERS, Marius M. SOLOMON et François SOUMIS. The VRP with Time Window. Cahiers du GERAD C99-13, 1999.

[CDG99] David CORNE, Marco DORIGO et Fred GLOVER, éditeurs. *New Ideas in Optimization*. McGraw Hill, 1999.

[CDM92] Alberto COLORNI, Marco DORIGO et Vittano MANIEZZO. Distributed Optimization by ant colonies. In *Toward a practice of autonomous systems : proceedings of the first European Conference on Artificial Life (ECAL 91)*, pages 134–142, Paris, 1992. Elsevier Publishing.

[CE69] Nicos CHRISTOFIDES et Samuel EILON. An algorithm for the vehicle-dispatching problem. *Operations Research Quaterly*, 20 :309–318, 1969.

[ČER85] V. ČERNY. A thermodynamical approach to the travelling salesman problem : an efficient simulation algorithm. *Journal of Optimization Theory and Application*, 45 :41–51, 1985.

[CEW71] Nicos CHRISTOFIDES, Samuel EILON et C. D. T. WATSON-GANDY. *Distribution Management : Mathematical Modeling and Practical Analysis*. Griffin, 1971.

[CG90] In Chan CHOI et Donald GOLDFARB. Solving multicommodity network flow problems by an interior point method. In T COLEMAN et Q. LI, éditeurs, *Large-Scale Numerical Optimization*, Philadelphia, 1990. SIAM.

[CG96] Boris V. CHERKASSKY et Andrew V. GOLDBERG. Negative-Cycle Detection Algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996.

[CGL+02] Jean-François CORDEAU, Michel GENDREAU, Gilbert LAPORTE, Jean-Yves POTVIN et François SEMET. A Guide to Vehicle Routing Heuristics. *Journal of Operational Research Society*, 53(5) :512–522, 2002.

[CJL02] Yves CASEAU, Francois-Xavier JOSSET et Francois LABURTHE. CLAIRE : Combining Sets, Search and Rules to Better Express Algorithms. In *Theory and Practice of Logic Programming*, volume 2-6, 2002.

[CL02] Pierre CHARDAIRE et Abdel LISSER. Simplex and interior point specialized algorithms for solving non oriented multicommodity flow problems. *Operations Research*, 50(2) :260–276, 2002.

[CL03] Jean-François CORDEAU et Gilbert LAPORTE. The Dial-a-Ride Problem : Variants, Modeling Issues and Algorithms. *4 OR*, 1(2) :89–101, 2003.

[CMR94] Jérôme CHIFFLET, Philippe MAHEY et Valérie REYNIER. Proximal decomposition for multicommodity network problems. *Telecommunication Systems*, 3 :1–10, 1994.

[CMT81] Nicos CHRISTOFIDES, Aristide MINGOZZI et Paolo TOTH. State-space relaxation procedures for the computation of bounds to routing problem. *Networks*, 11 :145–164, 1981.

[CPX02a] Ilog CPLEX 8.0 Reference Manual (Interactive Program & Callable Library). "Documents HTML", 2002.

[CPX02b] Ilog Concert Technology 1.2 Reference Manual. "Documents HTML", 2002.

[CR98] Eddie CHENG et Jennifer Lynn RICH. A Home Health Care Routing and Scheduling Problem. Technical Report TR98-04, Rice University, 1998.

[CW64] G. CLARKE et J.W. WRIGHT. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12 :568–581, 1964.

[DEI04] Camil DEMETRESCU, Stefano EMILIOZZI et Giuseppe F. ITALIANO. Experimental analysis of dynamic all pairs shortest path algorithms. In *SODA '04 : Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 369–378, New Orleans, Louisiana, 2004. Society for Industrial and Applied Mathematics.

[DES02] Guy DESAULNIERS. Bus and Driver Scheduling in Urban Mass Transit Systems. In *Travel and Trasnportation Workshop*. Institute for Mathematics and its Applications, 2002.

[DFJ54] George B. DANTZIG, Delbert Ray FULKERSON et S.M. JOHNSON. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954.

[DFMN00] Camil DEMETRESCU, Daniele FRIGIONI, Alberto MARCHETTI-SPACCAMELA et Umberto NANNI. Maintaining Shortest Paths in Digraphs with Arbitrary Arc Weights : An Experimental Study. In *WAE '00 : Proceedings of the 4th International Workshop on Algorithm Engineering*, Lecture Notes In Computer Science, pages 218–229, 2000.

[DGD01] Alexandre DURET-LUTZ, Thierry GÉRAUD et Akim DEMAILLE. Generic Design Patterns in C++. In *6th USENIX Conference on Object-Oriented Technologies and Systems*, pages 189–202, 2001.

[DL91] Martin DESROCHERS et Gilbert LAPORTE. Improvements and extensions to the Miller-Tucker-Zemlin subtour elmination constraints. *Operations Research Letters*, 10(1) :27–38, 1991.

[DL00] Ray DEITCH et Shaul P. LADANY. The one-period bus touring problem : Solved by an effective heuristic for the orienteering tour problem and improvment algorithm. *EJOR*, 127 :69–77, 2000.

[DL02] Jacques DESROSIERS et Marco E. LÜBBECKE. Selected Topics in Column Generation. Cahiers du GERAD G-2002-64, Décembre 2002.

[DLL93] Moshe DROR, Gilbert LAPORTE et François V. LOUVEAUX. Vehicle Routing with Stochastic Demands and Restricted Failures. *ZOR - Methods and Models of Operations Research*, 37 :273–283, 1993.

[DMA03] Christophe DUHAMEL, Antoine MAHUL et Alexandre AUSSEM. Routing with Neural-Based QoS Constraints. In *INOC Conference*, pages 201–206, 2003.

[DPR97] Christophe DUHAMEL, Jean-Yves POTVIN et Jean-Marc ROUSSEAU. A Tabu Search Heuristic for the Vehicle Routing Problem with Backhauls and Time Windows. *Transportation Science*, 31(1) :49–59, 1997.

[DR59] George B. DANTZIG et J. RAMSER. The truck dispatching problem. *Operations Research*, 12 :81–91, 1959.

[DUH01] Christophe DUHAMEL. *Un Cadre Formel pour les Méthodes par Amélioration Itérative - Application à deux problèmes d'Optimisation dans les Réseaux*. Thèse de Doctorat, LIMOS, Université Blaise Pascal, Clermont-Ferrand, 2001.

[DUN95] Tim DUNCAN. Experiments in the use of Neighbourhood Search Techniques for Vehicle Routing. Technical Report 176, Artificial Intelligence Applications Institute, University of Edinburgh, 1995.

[DW60] George B. DANTZIG et Philip WOLFE. Decomposition Principle for linear programs. *Operations Research*, 8 :101–111, 1960.

[EG02] Matthias EHRGOTT et Xavier GANDIBLEUX, éditeurs. *Multiple Criteria Optimization - State of the Art Annotated Bibliographic Surveys*, volume 52 de *International series in Operations Research and Management Science*. Kluwer Academic Publishers, Boston, 2002.

[EPP] David EPPSTEIN. Online bibliography on the K Shortest Paths Problem. "http ://www.ics.uci.edu/ẽppstein/bibs/kpath.bib".

[FDG01] Dominique FEILLET, Pierre DEJAX et Michel GENDREAU. Traveling Salesman Problems with Profits : an Overview. *en soumission*, 2001.

[FEI01] Dominique FEILLET. *Problèmes de tournées avec gains : étude et application au transport inter-usines*. Thèse de Doctorat, École Centrale Paris, 2001.

[FG94] Virgílio J. M. FERREIRA FILHO et Roberto D. GALVÃO. A Survey of Computer Network Design Problems. *Investigacio Operativa*, 4(2) :183–211, 1994.

[FGK73] Luigi FRATTA, Mario GERLA et Leonard KLEINROCK. The flow deviation method : An approach to store-and-forward communication network design. *Networks*, 3 :97–133, 1973.

[FGT95] Matteo FISCHETTI, Juan José Salazar GONZÁLEZ et Paolo TOTH. Experiments with a multi-commodity formulation for the Symmetric Capacited Vehicle

Routing Problem. In *3rd Meeting of the EURO Working Group on Transportation*, pages 169–173, 1995.

[FH04a] Serge FDIDA et Gérard HÉBUTERNE, éditeurs. *Méthodes exactes d'analyse de performance des réseaux*. Hermès-Science Publications, 2004.

[FH04b] Serge FDIDA et Gérard HÉBUTERNE, éditeurs. *Méthodes heuristiques d'analyse de performance des réseaux*. Hermès-Science Publications, 2004.

[FJ78] Marshall L. FISHER et Ramchandran JAIKUMAR. A decomposition algorithm for large-scale vehicle routing. Technical Report 78-11-05, Department of Decision Sciences, University of Pensylvania, 1978.

[FJ81] Marshall L. FISHER et Ramchandran JAIKUMAR. A generalized assignment heuristic for vehicle routing. *Networks*, 11 :109–124, 1981.

[FJM97] Marshall L. FISHER, Kurt O. JÖRNSTEN et Oli B. G. MADSEN. Vehicle Routing Problem with Time Windows : two optimization algorithms. *Operations Research*, 45(3) :488–492, 1997.

[FPR99] Michael FORSTER, Andreas PICK et Marcus RAITNER. The Graph Template Library. sur le site `http://www.infosun.fmi.uni-passau.de/GTL/`, depuis 1999.

[FR] Paola FESTA et Mauricio G.C. RESENDE. Annotated bibliography of grasp. rapport technique disponible à l'adresse internet : "http ://graspheuristic.org".

[FR89] Thomas A. FEO et Mauricio G.C. RESENDE. A probabilistic heuristic for a computationnally difficult set covering problem. *O.R. Letters*, 8 :67–71, 1989.

[FR95] Thomas A. FEO et Mauricio G.C. RESENDE. Greedy Randomized Adaptative Search Procedures. *Journal of Global Optimization*, 6 :109–133, 1995.

[FRI03] Jeffrey E. F. FRIEDL. *Maîtrise des expressions régulières*. O'Reilly, 2 edition, 2003.

[FT88] Matteo FISCHETTI et Paolo TOTH. *An Additive Approach for the Optimal Solution of the Prize-Collecting Travelling Salesman Problem*, pages 319–343. In GOLDEN and ASSAD [GA88], 1988.

[GA88] Bruce L. GOLDEN et Arjang A. ASSAD, éditeurs. *Vehicle Routing : Methods and Studies*. Studies in Management Science and Systems, 16. Elsevier Science Publishers B.V. (North-Holland), 1988.

[GAS67] T. GASKELL. Bases for the vehicle fleet scheduling. *Operations Research Quaterly*, 18 :291–295, 1967.

[GC95] Bezalel GAVISH et Suk-Gwon CHANG. Lower Bounding Procedures for Multiperiod Telecommunications Network Expansion Problems. *Operations Research*, 43(1) :43–57, 1995. survey alain.

[GCF98] Bernard GENDRON, Teodor Gabriel CRAINIC et Antonio FRANGIONI. *Multicommodity Capacited Network Design*, chapitre 1. In SORIANO and SANSÓ [SS98], 1998.

[GCM+01] Teresa GOMES, José CRAVEIRINHA, Lúcia MARTINS, Ernesto MARTINS et Marta PASCOAL. An Algorithm for Calculating the K Most Reliable Disjoint Paths with a Maximum Number of Arcs. In *ESREL2001, European Safety and Reliability International Conference,*, Torino, Italy, Sept 16-20 2001.

[GEN95] Bernard GENDRON. Modèles et algorithmes pour les problèmes d'optimisation de réseaux avec coûts fixes. Cahiers du GERAD CRT-1995-21, 1995.

[GEO72] Arthur M. GEOFFRION. Generalized Benders Decomposition. *Journal of Optimization Theory and Applications*, 4(10) :237–260, 1972.

[GER73] Mario GERLA. *The Design of Store-and-forward Networks for Computer Communications*. PhD Thesis, UCLA, 1973.

[GG61] Paul C. GILMORE et Ralph E. GOMORY. A linear programming approach to the cutting-stock problem — Part I. *Operations Research*, 9 :849–859, 1961.

[GG63] Paul C. GILMORE et Ralph E. GOMORY. A linear programming approach to the cutting-stock problem — Part II. *Operations Research*, 11 :863–888, 1963.

[GG74] Arthur M. GEOFFRION et Glenn W. GRAVES. Multicommodity Distribution System Design by Benders Decomposition. *Management science*, 20(5) :822–844, 1974.

[GG78] Bezalel GAVISH et Stephen C. GRAVES. The Travelling Salesman Problem and Related Problems. Technical Report OR 078-78, Operations Research Center, MIT, 1978. révisé et renommé en mars 1981.

[GHJV95] Erich GAMMA, Richard HELM, Ralph JOHNSON et John VLISSIDES. *Design Patterns :Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GIR] GIRO Inc. "http ://www.giro.ca/". éditeur des logiciels HASTUS et GiroRoute.

[GJL+03] Ronald GARCIA, Jaakko JÄRVI, Andrew LUMSDAINE, Jeremy SIEK et Jeremiah WILLCOCK. A Comparative Study of Language Support for Generic Programming. In *Proceedings of the 18th ACM SIGPLAN Conference OOSPLA*, 2003.

[GJR84] Martin GRÖTSCHEL, Michael JÜNGER et Gerhard REINELT. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, 32(6) :1195–1220, 1984.

[GL97] Fred GLOVER et Manuel LAGUNA. *Tabu Search*. Kluwer Academic Publishers, 1997.

[GLO77] Fred GLOVER. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8(1) :156–166, 1977.

[GLO86] Fred GLOVER. Future paths fur integer programming and links to artificial intelligence. *Computers & Operations Research*, 5 :533–549, 1986.

[GLO89] Fred GLOVER. Tabu Search : part I. *ORSA Journal of Computing*, 1(3) :190–206, 1989.

[GLO90] Fred GLOVER. Tabu Search : part II. *ORSA Journal of Computing*, 2(1) :4–32, 1990.

[GLO99] Fred GLOVER. Scatter search and Path relinking. In CORNE et al. [CDG99], chapitre 19, pages 297–316.

[GLON04] Éric GOURDIN, Bernard LIAU, Adam OUOROU et Dritan NACE. Optimisation des réseaux de télécommunications. *Bulletin de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, (12) :8–12, Juin 2004.

[GLP] GNU Foundation, GNU Linear Programming Kit (GLPK). "à l'adresse internet : http ://www.gnu.org/software/glpk/glpk.html".

[GLS96] Michel GENDREAU, Gilbert LAPORTE et René SÉGUIN. A Tabu Search Heuristic for the Vehicle Routing Problem with Stochastic Deamnds and Customers. *Operations Research*, 44(3) :469–477, 1996.

[GM95] Michel GONDRAN et Michel MINOUX. *Graphes et Algorithmes*. Eyrolles, 3e edition, 1995.

[GP99] Luis GOUVEIA et Jose Manuel PIRES. The asymetryc travelling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints. *European Journal of Operational Research*, 112 :134–146, 1999.

[GSF87] Marie-Claude GAUDEL, Michèle SORIA et Christine FROIDEVAUX. *Types de données et algorithmes : Recherche, Tri, Algorithmes sur les graphes*, volume 2. INRIA, 1987.

[GSS80] Giorgio GALLO, Claudio SANDI et Claudio SODINI. An algorithm for the min concave cost flow problem. *European Journal of Operational Research*, 4 :248–255, 1980.

[GT89] Andrew V. GOLDBERG et Robert E. TARJAN. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM (JACM)*, 36(4) :873–886, 1989.

[GTA99] Luca Maria GAMBARDELLA, Éric TAILLARD et Giovanni AGAZZI. MACS-VRPTW : a Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Technical Report IDSIA-06-99, IDSIA, Lugano, Suisse, 1999.

[GUE99] Cyrille GUEGUEN. *Méthodes de résolution exacte pour les problèmes de tournées de véhicules*. Thèse de Doctorat, École Centrale Paris, 1999.

[HAN81] Pierre HANSEN, éditeur. *Studies on Graphs and Discrete Programming*. Annals of Discrete Mathematics, North Holland, 1981.

[HAN86] Pierre HANSEN. The steepest ascent mildest descent heuristic for combinatorial programming. présenté au "Congress on Numerical Methods in Combinatorial Optimization", Capri, Italie, 1986.

[HMS03] John HERSHBERGER, Matthew MAXEL et Subhash SURI. Finding the k Shortest Simple Paths : A New Algorithm and its Implementation. In *ALENEX 2003, 5th Workshop on Algorithm Engineering and Experiments*, Baltimore, Maryland, January 11 2003.

[HOC97] Dorit S. HOCHBAUM, éditeur. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[HOL92] John H. HOLLAND. *Adaptation in natural and artificial systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Bradford Book, 1992. 1st MIT Press ed edition (1975).

[HTW97] Alain HERTZ, Eric TAILLARD et Dominique de WERRA. Tabu Search. *E. AARTS and J. K. LENSTRA, eds, Local search in combinatorial optimization*, pages 121–136, 1997.

[JM99] Víctor JIMÉNEZ et Andrés MARZAL. Computing the K Shortest Paths : a New Algorithm and an Experimental Comparison. In JS Vitter et CD Zaroliagis, éditeurs, *Algorithm Engineering*, volume 1668 de *Lecture Notes in Computer Science*, pages 15–29. Springer-Verlag, 1999.

[JZ91] Ralph E. JOHNSON et Jonathan ZWEIG. Delegation in C++. In *Journal of Object-Oriented Programming*, volume 4-11, pages 22–35, 1991.

[KEN78] Jeffery L. KENNINGTON. Multicommodity flows : A state-of-the art survey of linear models and solution techniques. *Operations Research*, 26(2) :209–236, 1978.

[KGV83] Scott KIRKPATRICK, C. Daniel GELATT Jr et Mario P. VECCHI. Optimization by Simulated Annealing. *Science*, 220(4598) :671–680, 1983.

[KLB04] Imdat KARA, Gilbert LAPORTE et Tolga BEKTAS. A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for the capacited vehicle routing problem. *European Journal of Operational Research*, 158 :793–795, 2004.

[KLE75] Leonard KLEINROCK. *Queueing Systems, Volume I : Theory.* Wiley Interscience, New York, 1975.

[KLE96] Jon KLEINBERG. Single-source unsplittable flow. In *Proceedings of 37th IEEE Symposium on Foundations of Computer Science (FOCS'96)*, 1996.

[KM97] Niklas KOHL et Oli B. G. MADSEN. An Optimization Algorithm for the Vehicle Routing Problem with Time Windows based on Lagrangian Relaxation. *Operations Research*, 45(3) :395–406, 1997.

[KS77] Jeffery L. KENNINGTON et M. Adel SHALABY. An effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems. *Management Science*, 23(9) :994–1004, 1977.

[LAP92] Gilbert LAPORTE. The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3) :345–358, 1992.

[LAS70] Leon S. LASDON. *Optimization Theory for Large Systems.* MacMillan, New-York, 1970.

[LÜB03] Marco E. LÜBBECKE. Combinatorially simple Pickup and Delivery Paths. *Central European Journal of Operations Research*, 2003. à paraître.

[LD60] A. LAND et A. DOIG. An Automatic Method of Solving Discrete Programming Problems. *Econometrika*, 28(3) :497–520, 1960.

[LIN65] Shen LIN. Computer solutions of the traveling salesman problem. *Bell Systems Computer Journal*, 44 :2245–2269, 1965.

[LIP96] Stanley B. LIPPMAN. *Inside the C++ Object Model.* Addison-Wesley, 1996.

[LK73] Shen LIN et Brian W. KERNIGHAN. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 44 :498–516, 1973.

[LLSC98] Qizhang LIU, Hoong Chuin LAU, Dennis SEAH et Sidney CHONG. An Efficient Near-Exact Algorithm for Large-Scale Vehicle Routing with Time Windows. In *5th World Congress on ITS*, Korea, 1998.

[LN87] Gilbert LAPORTE et Yves NOBERT. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31 :147–184, 1987.

[LND85] Gilbert LAPORTE, Yves NOBERT et Martin DESROCHERS. Optimal Routing under Capacity and Distance Restrictions. *Operations Research*, 33(5) :1050–1073, 1985.

[LR81] Jan Karel LENSTRA et Alexander H. G. RINNOOY KAN. Complexity of Vehicle Routing and Scheduling Problems. *Networks*, 11, 1981.

[LS98] Gilbert LAPORTE et Frédéric SEMET. Classical Heuristics for the Vehicle Routing Problem. Cahiers du GERAD G98-54, 1998.

[MA03] Antoine MAHUL et Alexandre AUSSEM. Neural-based quality of service estimation in MPLS routers. In *Supplementary Proceedings of the International Conference on Artificial Neural Networks (ICANN'2003)*, pages 390–393, Istanbul, Turkey, June 2003.

[MAJ05] Ali Ridha MAJHOUB. *Méthodes polyèdrales*, chapitre 8. In PASCHOS [PAS05], 2005.

[MBB01] Philippe MAHEY, Abdelhamid BENCHAKROUN et Florence BOYER. Capacity and flow assignment of data networks by generalized Benders decomposition. *Journal of Global Optimization*, 20 :173–193, 2001.

[MH97] Nenad MLADENOVIĆ et Pierre HANSEN. Variable Neighborhood Search. *Computers and Operations Research*, 24 :1097–1100, 1997.

[MIN61] George J. MINTY. Solving Stady-State Non Linear Networks of Monotone Elements. *IRE Transactions on Circuit Theory*, pages 99–104, 1961.

[MIN81] Michel MINOUX. *Optimum synthesis of a network with non simultaneous multicommodity flow requirements*. In HANSEN [HAN81], 1981.

[MIN86] Michel MINOUX. *Mathematical Programming. Theory and Algorithms*. John Wiley & sons, 1986.

[MIN89] Michel MINOUX. Network synthesis and optimum network design problem : models, solution methods and applications. *Networks*, pages 313–360, 1989.

[MKL03] Snežana MITROVIĆ-MINIĆ, Ramesh KRISHNAMURTI et Gilbert LAPORTE. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B*, 2003. à paraître.

[MPS98] Ernesto MARTINS, Marta PASCOAL et José SANTOS. The K Shortest Paths Problem. Technical report, CISUC, 1998.

[MS89] David R. MUSSER et Alexander A. STEPANOV. Generic Programming. In *Lecture Notes in Computer Science*, volume 358, pages 13–25. Springer-Verlag, 1989.

[MS00] Brian Mc NAMARA et Yannis SMARAGDAKIS. Static Interfaces in C++. In *First Workshop on C++ Template Programming*, 2000.

[MTZ60] C.E. MILLER, Albert W. TUCKER et R.A. ZEMLIN. Integer programming formulations and traveling salesman problems. *Journal of the Association for Computing Machinery*, 7(4) :326–329, 1960.

[MV94] Yuri A. MERKURYEV et Vladimir L. VISIPKOV. Optimization Methods in Discrete Systems Simulation. In *Proceedings of the European Simulation Symposium on Simulation*, pages 104–110, 1994.

[NAS51] John Forbes NASH. Non-cooperatives games. *Annals of Mathematics*, 45 :286–286, 1951.

[OK71] Kenji ONAGA et Osamu KALUSHO. On feasibility conditions of multicommodity flows in networks. *IEEE Transactions on Circuit Theory*, CT-18 :425–429, 1971.

[OMG03] OMG. Unified Modeling Language : Superstructure (Version 2). sur le site `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`, 2003.

[OMV00] Adam OUOROU, Philippe MAHEY et Jean-Philippe VIAL. A survey of algorithms for convex multicommodity flow problems. *Management science*, 46(1) :126–147, 2000.

[OPP95] Norbert OPPENHEIM. *Urban Travel Demand Modeling : From Individual Choices to General Equilibrium*. Wiley Interscience, New-York, 1995.

[OR76] I. OR. *Travelling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evansten, Illinois, 1976.

[O'R02] Martin J. O'RIORDAN. Technical Report on C++ Performance. Technical report, International Standardization Working Group ISO/IEC JTC1/SC22/-WG21, 2002.

[OSM93] Ibrahim H. OSMAN. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41 :421–451, 1993.

[OUO95] Adam OUOROU. *Décomposition proximale des problèmes de multiflot à critère convexe. Application au problème de routage dans les réseaux de télécommunication*. Thèse de Doctorat, LIMOS, Université Blaise Pascal, Clermont-Ferrand, Novembre 1995.

[OWE95] Guillermo OWEN. *Game Theory*. Academic Press, 3rd edition, 1995.

[PAS05] Vangelis Th. PASCHOS, éditeur. *Optimisation combinatoire : concepts fondamentaux*. Hermès, Paris, 2005.

[PD98] Panos M. PARDALOS et Dingzhu DU, éditeurs. *Network Design : Connectivity and Facilities Location : Dimacs Workshop April 28-30, 1997*, volume 40 de *Dimacs Series*, New-York, 1998. Discrete Mathematics and Theoretical Computer Science.

[PKGR96] Jean-Yves POTVIN, Tanguy KERVAHUT, Bruno-Laurent GARCIA et Jean-Marc ROUSSEAU. The Vehicle Routing Problem with Time Windows Part I : Tabu Search. *INFORMS*, 8(2) :158–164, 1996.

[PR95] Jean-Yves POTVIN et Jean-Marc ROUSSEAU. An Exchange Heuristic for Routing Problems with Time Windows. *Journal of the Operational Research Society*, 46 :1433–1446, 1995.

[PS97] Stefano PALLOTTINO et Maria Grazia SCUTELLA. Shortest Path Algorithms in Transportation models : classical and innovative aspects. Technical Report TR-97-06, Dipartimento di Informatica, Università di Pisa, 1997.

[QUI05] Alain QUILLIOT. *Les Problèmes de Synthèse de Réseaux : Fondements et Méthodes*, chapitre 1. In PASCHOS [PAS05], 2005.

[RJB98] James RUMBAUGH, Ivar JACOBSON et Grady BOOCH. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley Professional, 1998.

[ROS00] David ROS. *Étude de réseaux haut débit via la simulation de modèles fluides*. Thèse de Doctorat, Institut National des Sciences Appliquées, Rennes, 2000.

[RR94] César REGO et Catherine ROUCAIROL. Le Problème de Tournées de Véhicules : Étude et Résolution approchée. Technical Report 2197, INRIA, 1994.

[RR96] G. RAMALINGAM et Thomas W. REPS. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms*, 21 :267–305, 1996.

[SBL03] Michela SPADA, Michel BIERLAIRE et Thomas LIEBLING. Decision-aid Methodology for the School Bus Routing and Scheduling Problem. 3^{rd} *Swiss Transport Research Conference (STRC) - Monte Verità - March 19-21*, 2003.

[SCK+01] Lazar SPASOVIC, Steven CHIEN, Cecilia KELNHOFER-FEELEY, Quiang HU et Ya WANG. A Methodology for Evaluating of School Bus Routing : A Case Study of Riverdale, New Jersey. Technical Report N01-2088, National Center for Transportation and Industrial Productivity, New Jersey Institute of Technology, 2001.

[SGI] Silicon Graphics Inc., SGI Services & Support, Standard Template Library Programmer's Guide. à l'adresse internet : "'http ://www.sgi.com/tech/stl".

[SL00] Jeremy G. SIEK et Andrew LUMSDAINE. Concept Checking : Binding Parametric Polymorphism in C++. In *First Workshop on C++ Template Programming*, 2000.

[SLL02] Jeremy G. SIEK, Lie-Quan LEE et Andrew LUMSDAINE. *The Boost Graph Library : User Guide and Reference Manual*. Addison-Wesley, 2002.

[SM91] Gary L. SCHULTZ et Robert R. MEYER. An Interior Point Method for Block Angular Optimization. *SIAM Journal of Optimization*, 1(4) :583–602, 1991.

[SS98] Patrick SORIANO et Brunilde SANSÓ, éditeurs. *Telecommunications Network Planning*. Kluwer Academics Publishers, Boston, 1998.

[T2C] Transports publics de l'agglomération clermontoise (T2C). à l'adresse internet : http ://www.t2c.fr.

[TLZO01] Kay Chen TAN, Loo Hay LEE, Qili L. ZHU et Ke OU. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15, 2001.

[TV02a] Paolo TOTH et Daniele VIGO. Models, relaxations and exact approaches for the capacited vehicle routing problem. *Discrete Applied Mathematics*, 123 :487–512, 2002.

[TV02b] Paolo TOTH et Daniele VIGO, éditeurs. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[Xpr] Dash's XPress-MP Suite. "http ://www.dashoptimization.com/".

[YC02] Shangyao YAN et Hao-Lei CHEN. A scheduling model and a solution algorithm for inter-city bus carriers. *Transportation Research Part A*, 36 :802–825, 2002.

[YK02] Takeo YAMADA et Harunobu KINSHITA. Finding all the negative cycles in a directed graph. *Discrete Applied Mathematics*, 118 :279–291, 2002.

[ZN98] F. Benjamin ZHAN et Charles E. NOON. Shortest Path Algorithms : An Evaluation using Real Road Networks. *Transportation Science*, 32(1) :65–73, 1998.

RÉSUMÉ

L'objectif de cette thèse est de proposer des méthodes et des outils pour l'aide à la conception stratégique de réseaux de transports publics en milieu urbain. Cette problématique, proche des problèmes fondateurs de la Recherche Opérationnelle, est toujours d'actualité et bénéficie du dynamisme engendré par le domaine des Télécommunications, notamment à cause de l'ouverture à la concurrence des différents marchés. Cette classe de problèmes peut être inscrite dans un cadre plus large : les problèmes de synthèse de réseaux de mobilité.

Dans un premier temps, nous proposons un état de l'art des problèmes de synthèse de réseaux ainsi qu'un panorama des différentes méthodes et outils classiquement utilisés pour la résolution de tels problèmes. Cela nous permet de formaliser la notion de problèmes de synthèse de réseaux de mobilité avec demande élastique, c'est-à-dire où les demandes d'accès au réseau varient en fonction de la qualité de service offerte par le réseau. Nous présentons différentes modélisations ainsi que des extensions possibles qui tentent de prendre en compte des critères plus réalistes : identification d'une ligne unique, prise en compte de temps d'attente ou de service, des modes de déplacement alternatifs... Ces modèles sont étudiés sur des instances de taille modeste à l'aide d'un solveur de programmes linéaires entiers mixtes. Nous proposons ensuite la résolution d'un de ces problèmes par des métaheuristiques classiques - GRASP et Tabou - afin d'obtenir de bonnes solutions sur des instances plus grandes. Dans cette optique, nous formalisons une description particulière d'un circuit baptisée "géodésique" où le circuit est décrit par la donnée d'un nombre restreint de points du circuit et d'un plus court chemin entre deux points consécutifs. La complexité du problème étudié est telle que nous avons également défini une approximation de la fonction objectif afin d'en accélérer significativement son évaluation.

La dernière partie de la thèse est consacrée à la mise en œuvre d'une méthode fondée sur le schéma de Benders pour la résolution de problèmes de synthèse de réseaux de mobilité. Cette approche fait intervenir la résolution d'un problème auxiliaire difficile, pour lequel nous proposons différentes heuristiques fondées sur la recherche de cycle augmentant. Bien que l'approche globale semble prometteuse, il reste nécessaire d'améliorer la robustesse et la précision de la résolution du problème auxiliaire.

En annexe, nous proposons une réflexion sur certaines difficultés courantes de développement logiciel pour la Recherche Opérationnelle, et nous formalisons des schémas de conception reposant sur la programmation générique qui visent à concevoir des composants logiciels à la fois flexibles et performants. Nous présentons aussi une méthode de couplage entre une méthode d'optimisation et la simulation, baptisée "couplage par enrichissement", qui diffère du couplage classique - l'optimisation de simulation - et permet d'intégrer plus de réalisme dans les modèles de synthèse de réseaux de mobilité.

Mots-clés : Transports publics urbains, synthèse de réseaux, demande élastique, programmes linéaires entiers mixtes, métaheuristiques, GRASP, Tabou, Benders, Simulation

ABSTRACT

The aim of this thesis is to propose methods and tools to help strategic network designing dedicated to public transportation in urban environment. These issues, close to fundamental problems in Operations Research, are still topical questions and benefit from the dynamism of Telecomunications activity, especially due to the opening of the market to competition. This class of problems can fit into a larger framework : mobility network design problems.

First, we propose to survey network design problems and the existing methods and tools classically used to solve such problems. This study allows us to formalize the concept of mobility network design problems with elastic demand, that is, where the demands to access the network vary according to the quality of service provided by the network. We present various modeling as well as possible extensions that attempt to take into account more realistic criteria : identifying a single line, considering waiting times or service delays, other travelling patterns, etc. These models are studied on small-size instances with a mixed integer linear programming solver.

Then, we propose to solve one of these problems with classical metaheuristics - GRASP and Tabu - to obtain good solutions on bigger instances. From this perspective, we formalize a particular route description, named "geodesic", where the route is defined by a limited number of points on the route and a shortest path between two consecutive points. The complexity of the studied problem is such that we also defined an approximation of the objective function in order to significantly accelerate its evaluation.

The last part of the thesis is dedicated to the implementation of a method based on Benders' scheme to solve mobility network design problems. This approach requires the resolution of a difficult auxiliary problem, for which we propose heuristics based on the search of an increasing cycle. Although the global approach seems promising, it remains necessary to improve the robustness and the accuracy of the auxiliary problem resolution.

In the appendices, we propose considerations on some recurring issues in software design for Operations Research, and we formalize design patterns based on generic programming that aim to design both flexible and efficient software components. We also present a coupling method between optimization and simulation, named "model enhancement", that differs from the classical coupling - simulation optimization - and allows to integrate more realism in the models for mobility network design.

Keywords : Urban public transportation, network design, elastic demand, mixed integer programs, GRASP, Tabu, Benders, simulation