Algorithms for optimizing shared mobility systems

Daniel Chemla

UNIVERSITÉ PARIS-EST

Thèse présentée pour obtenir le grade de

Docteur de l'Université Paris-Est

Spécialité : Mathématiques

par

Daniel Chemla

Ecole Doctorale : MATHÉMATIQUES ET SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

ALGORITHMS FOR OPTIMIZING SHARED MOBILITY SYSTEMS

Thèse soutenue le 19 octobre 2012 devant le jury composé de :

Tal Raviv	*Rapporteur*
Louis-Martin Rousseau	*Rapporteur*
Pierre Carpentier	*Examinateur*
Jean-Philippe Chancelier	*Examinateur*
Dominique Feillet	*Examinateur*
Frédéric Semet	*Examinateur*
Frédéric Meunier	*Directeur de thèse*
Roberto Wolfler Calvo	*Directeur de thèse*

Dedicated to my grandfathers,
Max Chemla and René Gligseliger

La revolucion es como una
bicicleta ; si deja de andar, se cae

Che Guevara

Acknowledgements

This PhD was possible thanks to the collaboration of my two advisors Frédéric Meunier at École des Ponts ParisTech and Roberto Wolfler Calvo at Université Paris 13. It received a grant R2DS from the Île-de-France region. I would like to thank my two advisors particularly for their support. I was really lucky to work with them and I really enjoyed it. You were supportive and always available when I needed it.

First of all, I would like to thank Louis-Martin Rousseau and Tal Raviv for their detailled reviews and for coming from abroad to attend my presentation. I would like to thank Pierre Carpentier, Jean-Philippe Chancelier, Dominique Feillet and Frédéric Semet for being part of this jury. During these three years I had the chance to work with other researchers. I would like to thank Aristide Mingozzi who helped me with column generation method and showed me a bit of Emilia Romagna. I would also like to thank Michal Tzur and Tal Raviv who received me in Tel Aviv. In all the visited laboratories I worked together with people I also would like to thank, such as Marianne Trigalo for her comments on my dissertation, Enrico and Guillaume for their help in the ROADEF 2010 Challenge, Antoine for his help and drive in his hippy car, Emanuel and Marco for their helps with SCIP, Zheng for her participation in the Branch-and-Cut algorithm, Roberto for inviting me to the basketball game, Iris, Dana and Reut in TAU for showing me around and Mor for his biking lessons, and Alexandra for the super Canadian TSP and her support and Noam for his computer access. At last I would like to give special thanks to my coworkers, Djamal for having taught me “bâ tâ sâ”, Houssame for his support and the invention of the “glandredie”, Bernat el mejoar and Paolo for their patience during the ROADEF 2012 Challenge, Thomas the best travel agency, Rachana my movie dealer, Pauline and Vincent for their comments. I would like to thank all the members of the CERMICS, LVMT and LIPN departments, especially Nathalie, Catherine and Sylvie for their help. This thesis would not have been possible without Peio who showed me the thesis announcement.

At last I would like to thank Louise, all my friends and my familly for their support. And to my mother who wanted one of her sons to be a physician, “docteur” in French, this misunderstanding has led me here.

Abstract

Bikes sharing systems have known a growing success all over the world. Several attempts have been made since the 1960s. The latest developments in ICT have enabled the system to become efficient. People can obtain real-time information about the position of the vehicles. More than 200 cities have already introduced the system and this trend keeps on with the launching of the NYC system in spring 2013. A new avatar of these means of transportation has arrived with the introduction of Autolib in Paris end of 2011.

The objective of this thesis is to propose algorithms that may help to improve this system efficiency. Indeed, operating these systems induces several issues, one of which is the regulation problem. Regulation should ensures users that a right number of vehicles are present at any station anytime in order to fulfill the demand for both vehicles and parking racks. This regulation is often executed thanks to trucks that are travelling the city.

This regulation issue is crucial since empty and full stations increase users' dissatisfaction. Finding the optimal strategy for regulating a network appears to be a difficult question. This thesis is divided into two parts. The first one deals with the "static" case. In this part, users' impact on the network is neglected. This is the case at night or when the system is closed. The operator faces a given repartition of the vehicles. He wants the repartition to match a target one that is known *a priori*. The one-truck and multiple-truck balancing problems are addressed in this thesis. For each one, an algorithm is proposed and tested on several instances. To deal with the "dynamic" case in which users interact with the system, a simulator has been developed. It is used to compare several strategies and to monitor redistribution by using trucks. Strategies not using trucks, but incentive policies are also tested: regularly updated prices are attached to stations to deter users from parking their vehicle at specified stations. At last, the question to find the best initial inventory is also addressed. It corresponds to the case when no truck are used within the day. Two local searches are presented and both aim at minimizing the total time lost by users in the system. The results obtained can be used as inputs for the target repartitions used in the first part.

During my thesis, I participated to two EURO-ROADEF challenges, the 2010 edition proposed by EDF and the 2012 one by Google. In both case, my team reached the final phase. In 2010, our method was ranked fourth over all the participants and led to the publication of an article [78]. In 2012, we ranked eighteenth over all the participants. Both works are added in the appendix.

Résumé court

Les systèmes de vélos en libre-service ont connu ces dernières années un développement sans précédent. Bien que les premières tentatives de mise en place remontent aux années 60, l'arrivée de technologies permettant un suivi des différents véhicules mis à la disposition du grand public et de l'état des bornes de stationnement en temps réel a rendu ces systèmes plus attractifs. Plus de 200 villes disposent de tels systèmes et cette tendance se poursuit avec l'entrée en fonctionnement du système de New York prévue pour mars 2013. La fin de l'année 2011 a été marquée par l'arrivée d'un nouvel avatar de ce type de transport avec la mise en place d'Autolib à Paris.

L'objectif de cette thèse est de proposer des algorithmes d'aide à la décision pour l'optimisation de réseaux de transport en libre-service. L'exploitation de ces systèmes, qui fleurissent actuellement un peu partout dans le monde, pose en effet de nombreux problèmes, l'un des plus cruciaux étant celui de la régulation. Cette dernière a pour objectif de maintenir dans chaque station un nombre de véhicules ni trop faible, ni trop élevé, afin de satisfaire au mieux la demande. Cette régulation se fait souvent par le biais de camions qui effectuent des tournées sur le réseau.

Il apparaît rapidement que la question d'une régulation optimale à l'aide d'une flotte fixée de camions est une question difficile. La thèse est divisée en deux parties. Dans la première partie, le cas "statique" est considéré. Les déplacements de véhicules dus aux usagers sont négligés. Cela traduit la situation la nuit ou lorsque le système est fermé à la location. L'opérateur doit redistribuer les véhicules afin que ceux-ci soient disposés selon une répartition définie. Les problèmes de rééquilibrage avec un ou plusieurs camions sont traités. Pour chacun des deux cas, un algorithme est proposé et utilisé pour résoudre des instances de tailles variées. La seconde partie traite du cas "dynamique" dans lequel les utilisateurs interagissent avec le système. Afin d'étudier ce système complexe, un simulateur a été développé. Il est utilisé pour comparer différentes stratégies de redistribution des véhicules. Certaines utilisent des camions se déplaçant dans la ville pendant la journée. D'autres tentent d'organiser une régulation intrinsèque du système par le biais d'une politique d'incitation : des prix mis à jour régulièrement encouragent les usagers à rendre leur véhicule dans certaines stations. Enfin, si on choisit de ne pas utiliser de camion durant la journée, la question de la détermination du nombre optimal de véhicules à disposer à chaque station se pose. Deux méthodes de recherche locale visant à minimiser le temps total perdu par les usagers sont présentées. Les résultats obtenus peuvent

servir pour la définition des répartitions cibles de la première partie.

Durant ma thèse, j'ai pu participer à deux challenges EURO/ROADEF, celui de 2010 proposé par EDF et celui de 2012 proposé par Google. Dans les deux cas, mon équipe a atteint les phases finales. Lors de l'édition de 2010, notre méthode est arrivée quatrième et a donné lieu à une publication [78]. En 2012, notre méthode est arrivée dix-huitième sur tous les participants. Les travaux menés dans ces cadres sont ajoutés en annexe.

Contents

Résumé de la thèse

Les systèmes de vélos en libre-service ont connu ces dernières années un développement sans précédent. Bien que les premières tentatives de mise en place remontent aux années 60, l'arrivée de technologies permettant un suivi des différents véhicules mis à la disposition du grand public et de l'état des bornes de stationnement en temps réel a rendu les systèmes plus attractifs. Plus de 200 villes du monde disposent de tels dispositifs et cette tendance se poursuit avec l'entrée en fonctionnement du système de New York prévue pour mars 2013. La fin de l'année 2011 a été marquée par l'arrivée d'un nouvel avatar de ce type de transport avec la mise en place d'Autolib à Paris.

Si les nouvelles technologies ont permis d'assurer plus de fiabilité dans la localisation des véhicules et de responsabiliser davantage les utilisateurs en traçant l'identité des utilisateurs, elles n'ont toutefois pas résolu le problème des déséquilibres qui peuvent survenir durant la journée. Il n'est en effet pas rare que des stations se retrouvent pleines ou vides pendant des périodes de temps plus ou moins longues. Ces évènements diminuent l'efficacité du système de transport, les utilisateurs se présentant à ces stations ne pouvant pas trouver de véhicules ou de places de parking selon le cas. Le second cas de figure est bien plus préoccupant : un utilisateur qui ne trouve pas de véhicule peut toujours emprunter un autre moyen de transport, alors qu'abandonner son véhicule expose à des pénalités financières. Si ces incidents se reproduisent trop souvent, les utilisateurs risquent tout simplement d'abandonner le système de transport partagé pour lui préférer d'autres moyens de transport plus fiables. Une des difficultés de ces déséquilibres est qu'ils peuvent survenir à différents endroits de la ville selon l'heure de la journée. C'est en effet une conséquence des mouvements pendulaires maison-travail qui ont lieu tous les jours : un quartier d'affaire souffrira d'un trop-plein de véhicules le matin lorsque les employés y arrivent, et d'une pénurie le soir lorsque ces mêmes employés souhaitent rentrer chez eux. La problématique inverse peut se produire dans les quartiers résidentiels.

Le but de cette thèse est de proposer des algorithmes permettant une meilleure gestion

d'un réseau de transport partagé. L'objectif de ces travaux est de limiter les évènements "stations pleines" ou "stations vides". Dans de nombreuses villes déjà, des investissements ont été effectués afin de parer à ces problèmes. Une meilleure gestion des moyens déjà déployés permettrait donc d'améliorer les résultats sans augmenter les coûts. Les problèmes de déséquilibre sont une des principales sources de non-renouvellement des abonnements, comme en témoignent de nombreux articles de presse, tel celui paru dans Le Figaro du 23 mars 2010 [12]. Albert Asseraf, directeur général stratégie, études et marketing de JCDecaux y déclare qu'"*il n'est statistiquement pas possible de trouver un vélo ou une place dans* 100% *des cas*". Si cela reste vrai, on peut toutefois essayer par divers procédés de conserver une bonne couverture de la ville, en évitant que des zones entières souffrent d'excédents ou de carences en véhicules. Dans le cas des systèmes de vélos en libre-service, plusieurs camions parcourent la ville et transportent des vélos. Dans le cas du système Autolib, plus d'une centaine d'employés déplacent les voitures. Ces activités peuvent se faire plus facilement la nuit du fait d'un trafic faible.

Les travaux présentés ici se regroupent en deux parties. La première partie qui regroupe les chapitres 2 et 3 traite de la problématique rencontrée par les opérateurs de ces systèmes la nuit : les véhicules sont répartis sur les stations, mais leur répartition peut être différente de la répartition identifiée comme étant la plus à même de répondre aux besoins de l'heure de pointe du matin. Dans ces deux premiers chapitres, cette répartition cible est une donnée. Le faible nombre d'utilisateur étant présent dans le réseau, leur impact sur le système peut être négligé. Le problème devient alors un problème de redistribution de véhicules des stations ayant un excédent de véhicules vers celles souffrant d'une pénurie de véhicules. La seconde partie regroupe les chapitres 4, 5 et 6. Elle porte sur les problèmes rencontrés en temps réel durant la journée lorsque l'utilisation du système est importante. Le Chapitre 6 traite de l'identification de la répartition permettant de minimiser le temps moyen perdu par les utilisateurs sur toute une journée du fait de ces déséquilibres. Cela peut permettre de proposer une répartition comme cible dans la première partie.

Le Chapitre 2 traite du problème de rééquilibrage d'un réseau avec un unique camion, problème que nous avons appelé le *Single-Vehicle One-commodity Capacited Pickup-and-Delivey Problem* (SVOCPDP). Pour chaque station, le nombre de véhicules présents est donné, ainsi que le nombre de véhicules souhaités. Cela permet de définir les stations *pickup* comme celles

ayant un excédent de véhicule, les stations *delivery* comme celles ayant une pénurie de véhicule et les stations *initially balanced* comme celles dont le nombre de véhicules initialement présents est égal au nombre de véhicules souhaités. Un camion se trouve initialement à un dépôt. Il a une capacité et peut donc transporter un nombre borné de véhicules en même temps. L'objectif est de trouver un parcours de coût minimal permettant de déplacer les véhicules des stations *pickup* vers les stations *delivery* afin d'atteindre en chaque station le nombre souhaité de véhicule, où le coût d'un circuit est la distance parcourue par le camion.

Dans la littérature, le *swapping problem* introduit par Anily et Hassin [6] présente des caractéristiques proches. Différents types d'objets sont disponibles à chaque nœud d'un graphe et d'autres objets sont demandés en ces mêmes nœuds. Deux objets ne peuvent se trouver simultanément à un même nœud. Un unique camion de capacité unitaire peut déplacer les objets un à un, pouvant les déposer à une station pour revenir les chercher plus tard : c'est la version *préemptive* du *swapping problem*. Celui-ci existe aussi dans une version *non-préemptive*, dans laquelle ces dépôts temporaires, appelés *drop*, sont interdits. Une propriété intéressante est que le camion visite au plus trois fois chaque nœud. Cette propriété est l'idée centrale des méthodes de résolution par algorithme de *branch-and-cut* proposées par Gendreau *et al.* pour le résoudre dans ses versions préemptives [20] et non-préemptives [47]. Le SVOCPDP n'a qu'un type d'objets – les véhicules – et des capacités sur chaque station ainsi que sur le camion. La propriété du *swapping problem* signalée ci-dessus n'est plus valide. Hernandez Pérez et Salazar González [66] ont introduit le *One-Commodity Pickup-and-Delivery Travelling Salesman Problem* (1PDTSP). Des objets d'un seul type se trouvent en certains nœuds d'un graphe et doivent être amenés vers d'autres nœuds par un unique camion avec capacité. Celui-ci doit rééquilibrer le graphe en suivant un cycle hamiltonien. Ils proposent des méthodes de résolutions heuristiques [68] et une méthode exacte avec un algorithme de *branch-and-cut* [67].

Le SVOCPDP est proche du 1PDTSP mais il n'impose pas que la solution soit un cycle hamiltonien. Les *drops* et les *splits* sont autorisés : la demande à un nœud peut être servie par plusieurs visites. De plus, les stations *initially balanced* peuvent ne pas être visitées. Un modèle exact est donné pour le SVOCPDP. Celui-ci requiert un grand nombre de variables et donc n'est pas soluble. Une relaxation de ce problème est donnée. Celle-ci peut être résolue grâce à un algorithme de *branch-and-cut*. Il est prouvé que vérifier si une solution de la relaxation est aussi solution du SVOCPDP est un problème $\mathcal{NP}$-complet. Toutefois, dans la majorité des cas, il y a une solution du SVOCPDP avec un coût proche voire égal au coût de la solution de la relaxation. Plusieurs coupes sont intégrées dans l'algorithme afin de renforcer la solution

linéaire. Une recherche tabou est utilisée pour trouver des solutions au SVOCPDP. Celle-ci peut se faire en considérant exclusivement la liste des stations à parcourir et non les opérations de chargement ou déchargement à mener à chaque arrêt. Un algorithme de flot maximum permet en effet de reconstruire ces dernières à partir de la suite ordonnée des stations. La recherche tabou est lancée à deux reprises, la première fois à partir d'une solution obtenue par une méthode gloutonne, la seconde fois à partir de la solution obtenue par le *branch-and-cut*. La méthode est utilisée sur des instances jusqu'à 100 stations et avec diverses valeurs pour les demandes à chaque station. Plusieurs capacités du camion sont testées. Les instances ont été obtenues à partir des instances du 1PDTSP. Des solutions optimales ou proches de la solution optimale sont obtenues en des temps de calcul raisonnables jusqu'à des instances de taille moyenne.

Le Chapitre 3 traite de la résolution du *Multiple-Vehicle Balacing Problem* (MVBP), version à plusieurs camions du SVOCPDP. Une flotte homogène de camions est disponible au dépôt. L'objectif est de donner à chaque camion une route à suivre, suite de stations à visiter avec des opérations de chargements ou déchargements à effectuer. Le coût d'une route est la distance totale parcourue par le camion. Le but est de rééquilibrer le réseau avec un coût minimal. Afin d'éviter un déséquilibre dans les attributions aux différents camions, les routes ont une taille limitée par un paramètre $Rmax$. Cela permet de s'assurer de répartir les tâches sur plusieurs camions : en effet, il peut être plus intéressant en terme de coût de n'avoir qu'un unique camion visitant les stations tandis que les autres resteraient au dépôt. Cette borne $Rmax$ marque une différence avec la version monovéhicule. Une seconde différence est l'introduction de la convergence monotone vers l'équilibre : une station *pickup* ne peut que voir baisser son nombre de véhicules ; inversement, une station *delivery* ne peut que recevoir des véhicules. Cette contrainte est ajoutée pour éviter des problèmes de coordination entre plusieurs camions où l'un d'eux arriverait à une station pour y prendre des véhicules qui ne sont plus là. Cela interdit les *drops*, possibles dans la version monovéhicule, mais conserve la propriété de *split*.

Dans la littérature, le problème le plus proche est le *Split Delivery VRP* (SDVRP) introduit par Dror et Trudeau [35]. L'objectif est de servir plusieurs clients à l'aide d'une flotte homogène de camions présente à un dépôt. Les clients doivent recevoir une certaine quantité d'un produit. Ils peuvent être servis par plusieurs camions : possibilité d'avoir des *splits*. Le but est de répondre à la demande des clients tout en minimisant la distance totale parcourue par l'ensemble des camions. Dror et Trudeau ont identifié certaines propriétés des solutions opti-

males. Archetti *et al.* proposent une recherche tabou [7] et un algorithme exact [9] qui permet de résoudre à l'optimum des instances jusqu'à une quarantaine de clients. La méthode proposée de résolution du MVBP est utilisée sur ces mêmes instances. Dans la littérature, on trouve aussi le *Pickup and Delivery Problem with Time Windows* (PDPTW) : des objets doivent être collectés à des endroits et distribués à d'autres, le tout en respectant des fenêtres de temps. Les points de collecte et de livraison sont couplés, alors que le MVBP est un problème *Many-to-Many* : tout véhicule peut aller de toute station *pickup* vers toute station *delivery*. De très nombreuses méthodes existent pour résoudre le PDPTW. Seule celle de Baldacci *et al.* [10] est citée ici car elle sert d'inspiration à celle que nous proposons pour la résolution du MVBP.

Avant de présenter un modèle pour résoudre le MVBP, certaines propriétés caractérisant les routes d'une solution optimale sont identifiées. Elles généralisent les propriétés du SDVRP. Ces propriétés appelées règles de dominance vont nous permettent de mieux décrire l'espace des solutions à considérer. Ainsi le nombre d'utilisations des arcs reliant deux stations du même type est borné ; de même, le nombre d'utilisations des arcs reliant des stations de types différents et tels que le camion n'est pas complètement plein ou complètement vide peut être borné. Un modèle *set partitioning* est ensuite explicité. Néanmoins sa taille étant exponentielle, une méthode de génération de colonnes est mise en œuvre afin de résoudre sa relaxation linéaire. La contrainte d'intégrité des variables de décision est relaxée et seul un sous-ensemble de routes est considéré : c'est le problème maître. Seules les routes susceptibles d'améliorer le coût du problème maître sont ajoutées dans le sous-ensemble des routes considérées. Leur identification se fait par la résolution d'un problème de *pricing*. La méthode utilisée s'inspire de celle de Baldacci *et al.* [10]. Des demi-routes aller et retour sont générées par un algorithme d'expansion utilisant des règles de dominances entre demi-routes et une borne de complétion. Cette borne est obtenue par programmation dynamique. Ajoutée au coût d'une demi-route en expansion, on obtient une borne inférieure sur le coût d'une route complète utilisant cette demi-route. Les demi-routes aller et retour compatibles sont ensuite associées afin d'obtenir des routes complètes. Celles dont le coût réduit est négatif sont ajoutées dans le problème maître. Lorsqu'il n'y a plus de route à coût réduit négatif, le problème maître est résolu à l'optimum et la valeur obtenue est une borne inférieure à celle du problème originel. Afin de rehausser cette borne inférieure, différentes inégalités valides sont ajoutées dans la formulation du problème maître. Certaines d'entre elles proviennent de l'utilisation de fonctions *dual feasible*. Ces fonctions éliminent des solutions du relâché mais pas de solution entière. Une revue sur ces fonctions a été réalisée récemment par Clautiaux

et al. [26]. Quatre fonctions *dual feasible* sont utilisées ici : deux d'entre elles proviennent de cet article, les deux autres sont *ad hoc*. D'autres inégalités introduites sont des inégalités de cliques. Elles reprennent les propriétés de dominance mentionnées précédemment. Les variables duales associées aux premières inégalités peuvent être introduites dans le calcul de la borne de complétion, mais pas les secondes. Une fois toutes les inégalités valides ajoutées, on obtient une nouvelle borne inférieure, plus proche de la solution entière. Afin d'obtenir une borne supérieure, un algorithme mémétique avec découpe optimale est mis en œuvre. Comme dans le cas monovéhicule, il est possible de ne considérer que les suites de stations et de reconstruire les chargements et déchargements à chaque arrêt par un algorithme de flot maximum, qui repose sur un graphe similaire à celui du cas précédent. La méthode de découpe optimale est inspirée de celle de Prins [73]. Les différentes routes sont juxtaposées pour former un grand chromosome. Les opérations de *cross-over* ont lieu sur ce grand chromosome. L'algorithme de flot maximum de la version monovehicule est alors utilisé sur les nouveaux chromosomes comme s'il s'agissait d'une unique grande tournée. Si une grande tournée n'est pas réalisable, alors quel que soit le découpage, on ne peut pas obtenir de routes permettant le rééquilibrage du réseau, donc le nouveau chromosome est abandonné. Sinon, on cherche à faire un découpage optimal en prenant en compte les distances parcourues grâce à un algorithme de programmation dynamique avec étiquettes. Ce découpage peut ne pas respecter la réalisabilité des routes, mais avec les mouvements de voisinage on la retrouve. Ces mouvements sont des 2-OPT au sein des routes et entre les différentes routes. Une fois une borne supérieure obtenue, on introduit toutes les routes qui sont dans le *gap*. Cela nous assure d'avoir la solution optimale du problème. Des résultats sont donnés pour des instances allant jusqu'à 40 stations pour un temps d'exécution limité à 6 heures. Les petites instances sont toutes résolues à l'optimum. Sur les instances de taille moyenne, on trouve parfois l'optimum et parfois on ne peut obtenir de garantie d'optimalité de la solution dans le temps imparti. Cela reste vrai sur les grandes instances. Sur les instances du SDVRP résolues par Archetti *et al.* [9], l'algorithme retrouve 5 des 6 solutions optimales.

La seconde partie traite des problèmes de déséquilibre qui peuvent intervenir durant la journée. L'évolution de la répartition des véhicules dépend fortement de l'utilisation des stations. Par exemple, les impacts sur le réseau de la saturation d'une station sont complexes à évaluer : un utilisateur souhaitant rendre un véhicule à une station pleine va chercher à le restituer dans une station voisine. L'écho d'un problème de saturation ou de pénurie de

véhicules à une station peut ainsi se propager à tout le réseau. Herbert Simon, Prix Turing 1975 et Prix Nobel en Économie 1978, définit les systèmes complexes [81] comme « un système [. . .] composé d'un grand nombre d'éléments qui interagissent de façon complexe ». Un système de transport partagé entre exactement dans le cadre de ces systèmes.

Afin de pouvoir l'étudier, le Chapitre 4 décrit le simulateur OADLIBSim. Celui-ci permet de simuler facilement l'évolution d'un tel système de transport et de tester différentes stratégies utilisant des camions et/ou des politiques incitatives afin d'améliorer la régulation des véhicules en temps réel. Plusieurs caractéristiques du réseau doivent être fournies au simulateur, comme les paramètres des lois gouvernant les temps nécessaires pour rejoindre une station à partir d'une autre suivant le moyen de déplacement, la matrice Origine-Destination (O-D) régissant les probabilités du choix des destinations pour un utilisateur arrivant à une station ou encore les taux d'arrivée des utilisateurs par station. Différents profils d'utilisateurs peuvent aussi être définis selon leur patience : alors qu'un utilisateur impatient quittera le réseau si la première station qu'il visite n'a pas de véhicule disponible, un utilisateur moins "nerveux" pourra explorer d'autres stations environnantes dans la limite d'un nombre de stations spécifié et d'un temps limite. Le même processus peut avoir lieu lorsqu'il s'agit de rendre un véhicule loué. Enfin, si le choix est fait de mettre en œuvre une politique de prix incitant les utilisateurs à rendre leur véhicule à certaines stations, des paramètres permettent de comparer le coût enduré par un utilisateur se rendant à une station différente de celle initialement choisie. Plusieurs types d'utilisateurs peuvent coexister à l'intérieur du simulateur, les proportions de chacun dans le nombre total d'utilisateurs devant simplement être spécifiées. Le simulateur propose différents indicateurs pour évaluer l'efficacité des différentes méthodes. Les utilisateurs sont recensés selon le service reçu par le système : le nombre d'utilisateurs ayant pu effectivement bénéficier du service dans la limite de leur patience, le nombre d'utilisateurs perdus par manque de véhicules disponibles et ceux perdus par manque de places aux stations visitées. Si des véhicules sont transportés, le ratio du nombre moyen d'utilisateurs satisfaits gagnés sur le nombre de véhicules transportés est aussi calculé. Dans le cas d'une politique incitative, le nombre d'utilisateurs préférant rendre le véhicule à leur station d'origine et effectuer le déplacement à pied est aussi fourni. Ce simulateur programmé en C++ est simple d'utilisation et peut être téléchargé sur le site internet du projet. Des méthodes de régulation temps réel peuvent aisément être ajoutées grâce à un *template* guidant leur implémentation.

Le Chapitre 5 présente différentes méthodes de régulations. Celles-ci ont été implémentées et comparées grâce au simulateur OADLIBSim. Avant de présenter ces méthodes, on prouve que trouver la stratégie permettant de maximiser la probabilité de capter le prochain utilisateur dans le cas ou il y a plusieurs stations vides est un problème $\mathcal{NP}$-complet. Et cela reste vrai si on s'intéresse aux deux prochains utilisateurs. Différentes stratégies heuristiques sont proposées. Toutes fonctionnent avec la définition d'un nombre cible de véhicules pour chacune des stations et ont pour objectif de maintenir le nombre de véhicules disponibles aux stations proches de ce nombre cible. Les premières heuristiques proposées utilisent un camion. Celui-ci reçoit comme mission d'aller dans les stations souffrant de déséquilibre afin d'y remédier. Les instructions reçues peuvent être de visiter une ou deux stations selon les cas. Afin d'évaluer les déséquilibres à venir, certaines heuristiques utilisent une prédiction du nombre de véhicules aux stations utilisant les taux d'arrivées des utilisateurs et la matrice O-D. Une autre heuristique recherche la politique optimale à mener afin que l'espérance de retour du système à l'équilibre soit minimale. Le système est décrit comme une chaîne de Markov avec un très grand nombre d'états. L'état tel que les stations sont toutes à l'équilibre peut être décrit comme l'état cible d'un problème de plus court chemin stochastique.

Ce problème est notamment traité dans le livre de Tsitsiklis et Bertsekas [17], où ils exposent un algorithme itératif de recherche de politique optimale. Cette méthode est appliquée ici. Elle nécessite toutefois une bonne connaissance de l'évolution de la chaîne de Markov. De façon à réduire son nombre d'états, les stations sont regroupées par *cluster*. Le comportement des utilisateurs décrit précédemment est rationnel. Dans le cas où un utilisateur arrive à un station pour louer un véhicule et que celle-ci est vide, mais qu'il y a une station pleine à côté, la pénibilité engendrée par ce déplacement peut être négligée. Cela permet de ne plus considérer toutes les stations mais seulement un nombre réduit de *clusters*. Par ailleurs, le nombre exact de véhicules présents au sein d'un *cluster* peut aussi être négligé. Un *cluster* sera équilibré si le nombre de véhicules présents permet d'avoir en moyenne des véhicules et des places disponibles à chaque station. Si le nombre de véhicules est faible, le *cluster* peut souffrir d'une pénurie de véhicules. Inversement, si ce nombre est grand, une surabondance de véhicules peut entraîner des difficultés pour trouver une place pour les utilisateurs souhaitant rendre leur véhicule. Trois zones sont ainsi définies : déficit de véhicules, équilibre, excès de véhicules. Les décisions quant aux déplacements des camions sont faîtes en regardant l'état de chaque *cluster*. Le nombre d'états ayant été réduit, le simulateur OADLIBSim est utilisé afin d'approcher la matrice de changement d'état de la chaîne de Markov.

Une stratégie n'utilisant pas de camion mais un système de prix sur les stations est aussi présentée. Un prix est associé à chaque station . Un utilisateur souhaitant s'y rendre devra s'acquitter de ce prix. Ces prix mis à jour régulièrement servent à dissuader les utilisateurs de se diriger vers les stations déjà fortement occupées pour leur indiquer d'autres stations ayant plus de places disponibles. Le problème de reroutage des utilisateurs est modélisé comme un problème de transport de Monge [61]. Le choix d'utiliser les variables duales de ce problème linéaire pour en faire les prix associés à chaque station permet d'obtenir de bons résultats.

Ces différentes stratégies sont comparées sur un jeu d'instances simulées sur des réseaux réalistes avec un nombre de stations allant de 20 à 250. L'amélioration obtenue par la mise en place de ces stratégies est visible en comparant les performances avec la version sans aucune stratégie temps réel. Les résultats indiquent que dans tel un système évoluant très rapidement, les stratégies à court terme obtiennent de meilleures performances que celles à long terme. Cela peut être déduit en comparant les résultats des différentes méthodes, bien que la méthode de recherche de politique optimale, gourmande en temps de calcul, pâtit aussi de la faiblesse de l'évaluation de la matrice de passage des états de la chaîne de Markov. Par contre, la prise en compte de la prédiction permet d'améliorer les résultats obtenus par rapport aux mêmes méthodes ne se fiant qu'à l'état courant du réseau pour prendre des décisions. Les meilleures performances sont obtenues par la méthode de prix, bien qu'un nombre non négligeable d'utilisateurs préfère ne pas utiliser le système une fois les prix affichés. C'est une des limites de la modélisation à demande inélastique qui suppose que tous les utilisateurs se présentant à une station utilisent effectivement le système.

Le Chapitre 6 aborde la question de la détermination du nombre de véhicules à disposer à chaque station. L'objectif est de trouver la répartition qui minimise le temps moyen perdu par les utilisateurs du fait de problèmes de déséquilibre : un utilisateur se présentant à une station vide pour louer un véhicule devra aller à pied à une autre station afin de trouver un véhicule disponible. Et le même mécanisme se produit pour un utilisateur arrivant à une station pleine souhaitant rendre son véhicule. Le temps perdu peut être mesuré comme la différence entre le temps effectivement mis par un utilisateur pour faire un déplacement et le temps minimal idéal qu'il aurait put mettre si le système était infaillible, le temps de trajet direct de sa station d'origine à sa destination avec le véhicule loué. Dans ce chapitre, on suppose qu'il y aucune politique de régulation temps réel durant toute la période de simulation. Cela peut être le cas dans un système comme Autolib où les activités de repositionnement ont lieu principalement

la nuit, bien que durant la journée la régulation ait aussi lieu mais en moindre mesure. Les taux d'arrivée à chaque station dépendent ici de la tranche horaire mais pas la matrice O-D. La modification d'une unité du nombre de véhicules initialement disposés à une station peut permettre de parer au premier évènement de pénurie ou d'excès en véhicules qui se produit à la station. Deux différentes méthodes de recherche locale utilisent ce constat. La première conserve le temps perdu à chaque station du fait des problèmes d'excès ou de déficit. La seconde s'intéresse au nombre d'occurrences de ces problèmes. Une fois un grand nombre d'itérations réalisé, le bilan est fait et le nombre de véhicules présent au début de la journée peut être modifié en conséquence. Les deux méthodes sont testées sur des données d'utilisation d'une grande ville américaine, perturbées afin d'obtenir plusieurs instances. Les deux méthodes sont lancées à partir de plusieurs solutions initiales dont l'une est le résultat fourni par l'algorithme de Raviv et Kolka [74] qui cherche à minimiser le nombre d'utilisateurs non satisfaits sur chaque station, sans prendre en compte le report d'utilisateurs d'une station sur ses voisines. Les résultats témoignent d'une robustesse des deux méthodes de recherche qui arrivent à des résultats comparables, indépendamment de leur point de départ.

Chapter 1

Introduction

1.1 Bike sharing system

The urban population increases. In 2009, the level of world urbanization has crossed the 50% line and this movement keeps going on. In the annual report published by the Department of Economic and Social Affairs of the United Nation [65], the level of world urbanization is expected to grow up to 68.7% in 2050. In what is called the "most developed regions", the rural population has been decreasing for 60 years. This trend is expected to start also in the "less developed regions" after 2025. Meantime, the growth rate of the urban population is positive in both types of regions, though bigger in the later. Governments can try to weaken this imbalance by using decentralization of administrations or by encouraging people and activities to move to rural regions. But effects would only be marginal and the number of mega-cities – metropolitan area with more than 10 billions of inhabitants – is expected to reach 29 in 2050.

As cities will extend themselves, inhabitants will have to cover larger distances to go to the cities centers, their workplaces or leisure facilities. Easy transportation access will become more and more an issue. It is already one in most of the big cities. Zavitsas *et al.* [88] gather data obtained from 16 cities, most of which are in Europe. They show that economic prosperity is related to efficient urban transportation system and denote problems raised by urban transportation.

One of them is the congestion problem. Congestion has clearly a negative impact as it diminishes the capacity of the existing network. It increases transports emissions as well as the energy-consumption per kilometer and heightens noise pollution. The increasing number of cities with high population density will enhance congestion problems. The authors show that

the average traffic speed in the Greater London has been decreasing over the last decades, and this is not an isolated case. Congestion is partially explained by people commuting to work or to other activities. This appears clearly as the average traffic speed is even slower during morning and evening peaks. Commuters' displacements provoke these peaks during which the number of kilometers of traffic jams can be exceptionally huge.

Furthermore, environment preoccupations appeared at the end of last century. Here again, transportation activities play a major role as they are responsible for a quarter of greenhouse gas emissions of the 27 European Union (EU) members states. Moreover, although overall emissions have diminished between 1990 and 2006, emissions due to transportation activities have increased by 27% over this period. In 2009, EU countries agreed on a prospect to reduce their greenhouse gas emission in 2020 by at least 20% below 1990 levels. This commitment cannot be fulfilled without a cut in transportation emissions.

For all these reasons, efficient means of transportation have to be introduced in order to limit today's congestion and be ready to absorb the arrival of new inhabitants. An increase in the use of personal car would not be an efficient solution as it would worsen the two former problems. EU countries are bound to respect their greenhouse reduction engagements. The environment-friendly Bike Sharing System (BSS) could help. If it cannot replace public transportation, it work as an incentive to stimulate people to change their ground transportation habits. And besides being a green transportation mean, it offers a suitable answer for part of inner-city transportation demand and counters the "last-kilometer problem". This expression refers to the distance between home and the public transportation system, a distance that can be too far to walk but could be covered by biking. This mean of transportation is not new, however it has a growing success all over the world. Several attempts have been made since the 1960s. Shaheen *et al.* [80] and DeMaio [31] present a study on this mean of transportation over time and divide the experiments into three generations. The first generation gathers systems, in Europe mainly, where everything is free. People can use bikes and leave them once arrived at their destination in the city centers. No stations are installed. The first realization of this system was done in Amsterdam, Holland, in 1968. Bikes have special distinctive signs such as their color – they usually are painted with one bright color: white in Amsterdam, Holland; yellow in La Rochelle, France; green in Cambridge, UK. But within a few months, most of the bikes are stolen and the others are vandalized. Almost all the experiments have failed. In La Rochelle, France, the system introduced in 1974 met success. The system is still running though slight

modifications were introduced into the offer in order to trace users [16]. However, this is the first success of bike sharing scheme in Europe. The second generation of systems includes bikes racks. To rent a bike, a small cash deposit has to be done at the station – deposit that is given back to the user when he parks the bike back at a rack. This coin-deposit system was first introduce in Copenhagen, Denmark, in 1995. The deposit was about USD 3. This system is a lot more reliable than its predecessor. However it is much more expensive as it request stations equipments. But users are unidentified. The low price of the deposit did not deter theft and vandalism. As a result, it was canceled in most of the places where it was started, or modified to trace users. The third generation of BSS was enabled thanks to the development of new Information and Communications Technologies (ICT). Bikes are parked at stations racks. To rent a bike, users have to identify themselves by using smart technologies such as mobile phone, membership card, or bank card – the list is not exhaustive. They are charged for the time they use the bike, after a first period during which it is free. Increasing price encourages them to return the bike at a station once their trip is finished. If the bike is not given back, a high punitive cost is charged to the identified user. This generation meets a huge success. The Vélo'v system in Lyon, France, launched in 2005, popularized it. It is the first time a BSS is operated at a large scale: with $1'500$ bikes at its start, this number has gradually increased to reach $3'000$ bikes. Following Lyon, the French capital city launched its own system, Vélib', in summer 2007. It immediately met a great success with almost $200'000$ registered users and more than 26 millions locations of bikes within the first year. These successes have enhanced this trend and BSSs have appeared in more than 200 cities. This movement keeps going on with the launching of a BSS in New York scheduled for spring 2013. The biggest BSS is in the city of Hangzhou, China, with about $2'400$ stations and up to $60'000$ bikes.

However, in Paris and some other cities, the system suffers from severe vandalism. Moreover, people often find themselves unable to either rent a bike or park it at station because there is no bike parked or no rack available. These issues led to a decrease in the number of users as stated in "Le Figaro" article [12] in 2008 about the Parisian Vélib' system. In Brussels, the Villo! system that opened in 2009 experienced imbalance problems and the press reported them [77]. A website was created by users to measure imbalances in the network (http://www.wheresmyvillo.be).

To reverse this trend, operators engaged themselves to minimize vandalism by reinforcing the equipment and to prevent stations from being full or empty by using real time regulation.

In the Vélib' system, the contract that engaged JCDecaux and the city of Paris was modified to include regulation objectives. The demand for transportation can be asymmetrical within the day. This leads to an imbalance of bikes with areas where all stations are full while in others there is a shortage of them. A fleet of trucks is available and turns around the city, moving bikes from a place to another. But how to efficiently plan a regulation system ? How to manage this fleet ? What directions to give to trucks drivers ? Could the system be regulating itself using incentive policies for users ?

1.2 Research motives

Bike sharing problems are relatively new, but there is already an important literature, addressing them from various points of view. Lathia, Ahmad and Capra [56] adopt a statistical approach to discuss the performances of existing systems. Vogel and Mattfeld [84] gather and study data they obtained from the Vienna bike sharing system, and give a model that could be used to further expand the network. Lin and Yang [57] propose a model that gives a strategic planning of a BSS considering a service level requirement. One of the problems BSSs operators face is to balance the network. Depending on days or on locations, some stations or areas of the cities could gather a huge number of bikes, leaving no rack available for users to park. And in other locations, it is the opposite with few bikes available. Moving bikes to balance the network and avoid shortage of bikes or racks is a problem that could fit within the many-to-many pickup and delivery problems class. However, in the daytime, the network is evolving very fast: for example in Paris, there is about $110'000$ rentals per day in average [2]. The number of bikes present at a station may change during the time a truck drives from a station to another. The main reason for long-term subscribers not to renew their subscription is the regulation problem. Several works on that topic have appeared recently, some of which are cited in an article of "Le Temps" printed in 2011 [82]. Raviv *et al.* [76] propose several models and algorithms to solve a bikes repositioning problem. Their objective is to find the best repositioning that can be achieved by several trucks within time limits, neglecting the impact of users on the system. The satisfaction function introduced by Kolka and Raviv [74] is used to evaluate the quality of a repositioning. Rousseau *et al.* [27] propose to solve a dynamic public bike sharing balancing problem.

The aim of this thesis is to study efficient algorithms and methods that could be used in operating BSSs and more generally any shared transport system. The focus point is the imbal-

ances problems as it appears to be a major issue for users. The dissatisfaction of roaming to find a bike or a rack is obvious. The work is separated between the *static problem* and the *dynamic problem*. Static problem means that users effect on the system is neglected. It could be the case if the service is closed or overnight when the system is nearly idle as there is almost no users. Note here that several systems around the world are closed during a few hours every days for maintenance and regulation operations. For instance, in Denver, US, the system [3] runs from Mars to December, from 5a.m. to 12a.m. The objective is then to use one or several trucks to balance the system. The problems addressed in the first part belong to the Vehicle Routing Problems class, which is detailed in the next section. On the opposite, dynamic problem refers to the balancing problem when users are in the system and modify the number of vehicles at stations. The question that has to be addressed is how to direct the trucks drivers to have the system providing a good level of service to the users. This is the problem faced at daytime by the operators of these systems. In most of the cities having a BSS, trucks are driving through the city bringing bikes from stations to others. Another problem that popped up is to find the best initial distribution of vehicles to minimize users' dissatisfaction. Indeed, the trucks impact on the system efficiency can be marginal when a good repartition of the vehicles in the morning could help to reduce dissatisfaction.

1.3 Routing problems and bikes balancing problems

Routing problems gather operational problems faced in the management of distribution tasks. The field of transportation has been one of the major focus of the Operation Research (OR) community. The interest in this topic arose very early. At first, only the savings that can be achieved have motivated to better schedule transportation activities. Growing environment concerns have then enhanced the attractiveness of this field of research. The first and original problem studied is the well-known Travelling Salesman Problem (TSP). Signs of the TSP are found as early as the 1830s in Germany in [19], though this article contains no mathematical treatment. It was named after the problem faced by travelling salesmen who have to visit several cities and so to plan a closed trail – cycle. But to save time or money, they want to find the cycle that would be the *less costly* for them. The first trace of mathematicians' interests for the TSP are in the 1930s when it seems to be studied both in Germany and the US. It became in the 1950s and 1960s a very popular topic for researchers.

In its formal definition, the objective is to plan a closed trail visiting once several cities

while minimizing the total number of kilometers travelled. The distance matrix is given. In 1954, Dantzig *et al.* [29] presented the optimal cycle linking 49 American cities, one per state + Washington D.C.. They were the first to present a integer linear program and to propose a cutting plane method. This process was later extended with the introduction of branch-and-cut algorithms. In 1962, the society Procter and Gamble ran a contest for finding the optimal cycle reaching 33 American cities. With no surprise, American researchers in OR were listed among the winners. In 1962, Held and Karp [50] presented a dynamic programming approach that successfully solves small size instances. The TSP was proven to be $\mathcal{NP}$-hard in 1972 after Karp proved the $\mathcal{NP}$-completeness of finding an Hamiltonian cycle in a graph, where an Hamiltonian cycle in a graph is a closed trail that visits all vertices once. Since then, the size of instances optimally solved have increased to reach today optimal solution for almost $90'000$ cities. Huge size instances come from very-large-scale integration problem where a huge amount of transistors have to be taken to put them into chips.

Other Vehicle Routing Problems (VRPs) have been defined since the Fifties. More complex than the TSP, they enabled to model and to study different real life transportation problems where other constraints appear. These different features lead to the definition of variants of the TSP, some of them are reported below. The asymmetric TSP is the same problem but where the distance matrix is asymmetric. In the sequential ordering problem, there are precedence constraints on the cities: some cities have to be visited before others.

In the Capacitated Vehicle Routing Problem (CVRP), a fleet of capacitated vehicles is available and to each cities – or customers – a demand is attached. The objective is to find the collection of closed trail such that every customer is visited once and that his demand is satisfied without exceeding the vehicle capacity. The CVRP first defined in [30] is one of the most studied variant on the TSP. It is a generalization of the TSP. Indeed, the TSP corresponds to an instance with a single vehicle available at the depot which capacity is greater than the sum over the demand of all customers, the depot being the initial city of the tour.

Multiple variants exist on the CVRP or the TSP, adding complications on the tour of the vehicle. In the CVRP case, see [83] for a list of them. Here we give a list of some of them that are relevant with regard to this thesis.

– Associating time windows to customers restrains the visit to occur within time limits.
– Enabling the customers to be visited by several vehicles by splitting his demand.
– Having pickup or delivery operations at the vertices; objects have to be picked up or

delivered at vertices. It could be the same or different commodities.

Split means that the demands may be satisfied by several vehicles. The Split Delivery VRP (SDVRP) was first introduced by Dror and Trudeau [35]. As in the CVRP, each customer has a demand that has to be served by capacitated vehicles that are parked at a depot. But vehicles can serve only part of the demand of a customer. Dror and Trudeau proved that the split dimension of the problem could lead to huge savings in term of the objective function [34]. The split component of the problem increases the complexity of the algorithm [8]. However, they found a property that limits the number of solutions to consider. If the costs satisfy the triangle inequality, then there exists an optimal solution to the SDVRP where no two routes have more than one customer with a split delivery in common.

Having pickup and delivery requests at vertices leads to different problems depending on specific characteristics. Laporte *et al.* [15] sort the different Pickup and Delivery Problems (PDPs) that were introduced by different authors. They are sorted with respect to how pickup and delivery vertices are paired or not. Here we will focus on the many-to-many problems. Many-to-many means that pickup and delivery points are not paired. An object picked up at a vertex can be delivered to any other vertex that has a demand for this type of object. Three different problems are in this class.

In the swapping problem, a single vehicle has to move unitary objects from a vertex to another. There are m different types of objects and the vehicle has a unitary capacity: it can only move one object at the time. Each vertex is associated with the type of object currently present at it, if any, and the desired object type, if any. For each type of object, the total demand is assumed to be equal to the total supply. The objective is to find a tour for a unique unit capacity vehicle at the end of which all objects are brought at a vertex where there is a request for an object of their type. This problem was introduced by Anily and Hassin in [6]. They showed that a vertex could be visited at most three times in the optimal solution. The problem has a *preemptive* version and a *non-preemptive* version. In the preemptive version, an object can be dropped at a vertex that is not its destination vertex to be collected later on by the vehicle when it comes back.

The second problem in this category is the One-Commodity Pickup-and-Delivery Travelling Salesman Problem (1PDTSP). It was first introduced by Hernández-Pérez and Salazar-González in [66]. Vertices are divided into pickup and delivery vertices. There is only one type of commodity. To each vertex is associated a non-zero demand, that is the amount of the

commodity to either pickup or deliver at the corresponding customer, respectively to its sign. A capacitated vehicle is given at a depot. The objective is to find the minimum Hamiltonian tour that visits once each customer and brings the commodities from pickup vertices to delivery ones. This Hamiltonian tour has to respect the capacity of the vehicle.

The Q-Delivery Travelling Salesman Problem (Q-DTSP) is a special case of the 1PDTSP where all the demands are unit demands. In this case, the demand at pickup or delivery vertices is only one unit of the commodity. It was introduced before the latter in 1999 by Motwani and Chalasani in [63]. Here again, the solution is an Hamiltonian tour. They propose an approximation algorithm to solve it. The same problem was studied at the same time by Anily and Bramel. In [5] they introduce the Capacitated Travelling Salesman Problem with Pickup and Delivery (CTSPPD) as an extension of the swapping problem with one commodity and a capacitated vehicle.

Balancing a shared transport system enters clearly in the many-to-many pickup and delivery problems class. Vehicles have to be moved from stations with an excess of vehicles to stations with a shortage of them. At first glance, the 1PDTSP could model this problem. But there is no reason for not allowing multiple visit at stations. For instance, assume that there is a pickup station where 6 vehicles need to be taken out and that a truck is nearby but has room for only 3 vehicles. Then it could load 3 bikes and come back later to load the others 3. In addition, central stations may be used as buffer where vehicles can be dropped to be recollected later. The 1PDTSP does not allow such operations to occur as its solution is required to be an Hamiltonian cycle. A consequence is also that unfeasible instances of the 1PDTSP could be solved if multiple visits were authorized.

In real time, there are additional constraints on the ways to manage the system. First, the number of bikes changes within the system; second, the computational time of the methods has to be very short to enable to quickly obtain instructions. The VRPs mentioned formerly deal with the static situation. To deal in real time with the dynamic problem faced by a shared transport system operator is another problem. The literature is less developed than in the static case. However, several works have been done on the topic. George and Xia [48] propose a modeling using closed queuing networks. Having an exact model of the system is not an easy task. The latest work using queuing model are done by Fricker and Gast [44]. Pesach *et al.* [71] propose an dynamic programming algorithm to find the best actions in order to minimize the

number of unserved users. They use a rolling horizon and launch their program regularly. The work of Contardo *et al.* [27] formerly mentioned aims also at minimizing the sum of unserved users. They introduce a time-discretized model of the system and use columns generation to obtain in short time instructions to give to the trucks. Here again the method is run regularly.

1.4 Thesis overview

This thesis is organized in two parts and five chapters followed by a chapter containing concluding remarks. Part I consists of chapter 2 and 3 and deals with the static problems. Part II includes chapter 4, 5 and 6 and deals with with the dynamic problems.

In Part I, the static problem is introduced. It is a many-to-many PDPs where the demand at vertices can be split. As stated before, authorizing multiple visits at stations could lead to huge savings in term of the objective function, but it increases the complexity of the problem. Two static problems are formally presented. In this part, the problem is to balance the system during the night with one or several capacitated trucks. These problems suit more to a BSS than a car sharing system. The commodity to move is called *bike*, and the word *vehicle* refers to the truck. The Single-Vehicle One-Commodity Pickup and Delivery Problem (SVOCPDP) is the single-vehicle version of the balancing problem. Bikes are initially distributed among the vertices of a graph. To each vertex is associated its *initial state*, and its *target state*, where state indicates the number of bikes parked at the vertex. A preliminary discussion with theoretical results such as special polynomial cases or approximation algorithms can be found in the paper by Benchimol *et al.* [14]. Three types of stations appear; stations which initial state are strictly lower (respectively greater) than target state are called *pickup* (respectively *delivery*) stations; stations which initial state is equal to target state are called *initially balanced* stations. A capacitated vehicle aims at redistributing the bikes in order to reach a target states distribution. Each vertex can be visited several times and can be moreover used as a buffer in which bikes are stored for a latter visit. The last property allows drops. This many-to-many PDP gathers features from both the 1PDTSP and the swapping problem. The second static problem that is introduced is a multiple-vehicle variant of the latter. It is called Multiple-Vehicle Balancing Problem (MVBP). A fleet of capacitated vehicles is available at a depot. Bikes are distributed over all stations. A target state is given for each station. In the MVBP case, convergence to the target distribution is *monotonous*: only loading (resp. unloading) operations can occur at

pickup (resp. delivery) vertices. This last property forbids drops to occur. The objective is to find the set of instructions to give to vehicle drivers that bring the system to its target state while dispatching the tasks to be done over several vehicles and while minimizing the total distance driven.

In Chapter 2 the SVOCPDP is studied. Its differences with the swapping problem and the 1PDTSP are explained in detail. A first exact arc-oriented mixed integer programming formulation (MIP) is given. This MIP introduces four sets of variables, some of which are indexed with four indices. The linear relaxation of this model could be very weak and therefore useless, and moreover the model can be intractable. Two consecutive relaxations of the problem are then proposed. They are proven to be equivalent. The meaning of this relaxation is explained, showing that it is a good relaxation, as in most experimental cases, an optimal solution of the relaxation is a solution for the original problem. Several results on the $\mathcal{NP}$-completeness of deciding whether a solution of the relaxation is a solution of the original problem are given. But the last relaxation needs only one set of integer variables, which are the number of times an arc is used. The interest of solving this relaxation appears then clearly. This is solved thanks to a branch-and-cut algorithm. For the branch-and-cut, three separation procedures are used. An upper bound of the optimal solution of the problem is obtained by a tabu search algorithm [49], which is based on some theoretical properties of the solution, once fixed the sequence of the visited stations. It is proven that a solution can be rebuilt knowing only the sequence of visited vertices: bike load can be found using a max-flow algorithm. Computational results are then given for instances with up to 100 vertices, and different capacities for the vehicle. Tests are also done on instances such that the demand at some stations can exceed the vehicle capacity.

Chapter 3 is devoted to the MVBP. Its differences with other pickup and delivery VRPs are emphasized. Several dominances rules and properties of the optimal solutions of the problems are proven. A set partitioning-like model with binary variables is given on the set of all the routes. A route is a tour starting from the depot visiting a subset of vertices with logistic operations to handle at each stop and ending at the depot. The fact that such a model with binary variables exists is not obvious. We can prove that the model is still exact thanks to the dominances aforementioned. However, its linear relaxation is solved with a column-and-cut generation algorithm, since its size being exponential in the size of the instances. The

relaxation is solved on a subset of variables also called columns. A *pricing subproblem* is solved to add columns that may improve the cost of the linear relaxation. For that purpose, a two-phase method using dynamic programming is explained. When no more negative reduce cost column can be added, cuts are added to enhance the linear relaxation of the original problem. Once neither cuts nor routes are to be added, a lower bound on the original problem is obtained. A memetic algorithm [62] provides an upper bound on the problem. Then, we add into the subset all columns candidate for being in the optimal solution. They are identified thanks to their reduced cost value. In that case, the original model solved on the subset of routes gives the optimal solution of the original problem. Results are given for instances with up to 40 stations.

Part II deals with the dynamic problem. It refers to the balancing problem in real time. It models the real life problem faced by the operators in the daytime while the system is open. This part is relevant for any shared transport system, so the word *vehicle* refers this time to the commodity that is shared – bikes in a BSS, cars in a car sharing system – while the word *truck* is used to represent the regulation activities. In this case, decisions have to be taken regarding balancing vehicles in an uncertain environment. For that purpose, a simulator of shared transport system is described. It models users' actions on the system. This simulator is used to compare different strategies designed for improving users' satisfaction. Some of them give instructions to truck driving around the city on where to go and how many vehicles to load or unload at stations. Other strategies use incentives to have the system regulating itself without any truck. Another problem that is studied is the Initial Inventory Problem (IIP). In the IIP, the objective is to find the number of vehicles to deploy in the city and where to deploy them in order to minimize the time users would "lose" using the system, if no regulation activities were done during a time period – a day, a morning.

Chapter 4 describes the simulator OADLIBSim that was developed to model users' actions in a shared transport system. At first, this mean of transportation is described with its model. The time needed by a vehicle, a truck or a pedestrian to go from vertex i to vertex j are random variables. Users arrive at stations with respect to a Poisson law whose parameters are given. They choose their destination with respect to an O-D matrix. In case users do not find any vehicle at their depart station or any parking place at their destination one, they roam the nearby stations knowing the number of vehicles parked there. Different types of users

can be defined, with different acceptance thresholds for roaming after which the user leaves the system unsatisfied. Users willingness to pay for modifying their destination can also be precised in the case of a pricing strategy. Thanks to this simulator, different indicators can easily be computed to estimate the system efficiency.

Chapter 5 presents different real time strategies that have been tested to improve the level of service of a shared transport system. These methods are heuristics and are divided into those using trucks for regulating the system in real time and those using incentives to have users parking their vehicles at a different station from their initial destination. Three main methods are presented. The first one simply sends trucks at most unbalanced stations to try to regulate them. There are several ways for evaluating the imbalances, leading to different algorithms using either the current number of vehicles or the forecast number of vehicles. This forecast can be calculated using the O-D matrix and the mean number of users showing up at stations. The second method uses an optimal policy algorithm [17] and runs off line. Its results are used to give instructions to trucks drivers. The latter method needs a good knowledge of the system and its behaviour. The last method uses an incentive strategy, associating prices to each station. Knowing their destination and the prices, users can chose either to go to their destination or to a nearby station and pay the price to park there, or to leave the system and walk to their destination. However, modifying their destination does not ensure them to find an available parking place once they get to the station. The prices associated to stations are found thanks to a linear model. Its translation into money is done using the mean value associated to one hour in Western countries.

Chapter 6 deals with another problem that is finding the IIP. In the IIP, a network is given with its stations and their capacities. Users are showing up at stations with respect to a Poisson law which parameters change over time. They choose their destination with respect to a given O-D matrix. The ideal time they would spend in the simulator is the time for a vehicle to go from their origin station to their destination. However, because of capacity issues, they may have to roam for finding a vehicle or an available parking place, leading to a time loss. The objective is to find how many vehicles to put at each station before the simulation starts in order to minimize the total time lost by all users. For that purpose, two local searches are outlined and compared. Both use the simulator OADLIBSim to find at which stations problems occur and to modify the initial number of vehicles in an appropriate way.

Part I

The static problem

This part gathers works done in collaboration with Frédéric Meunier and Roberto Wolfler Calvo.

Chapter 2 has an article version [24] that was submitted to Discrete Optimization and was accepted up to minor revisions.

Chapter 3 was presented at ROADEF 2012 and ODYSSEUS 2012 conferences. A journal version of this chapter is in preparation [23].

Chapter 2

Solving the Single-Vehicle One-commodity Capacitated Pickup and Delivery Problem

2.1 Introduction

The Single-Vehicle One-commodity Capacitated Pickup and Delivery Problem (SVOCPDP) represents the problem faced by an operator of a BSS during the night when the number of moving bikes is negligible and when the city is divided into districts. Each district is covered by a single vehicle that has to redistribute the bikes in order to respond to the morning peak at best. To get an idea of the size of such a system, the numerical features of the Vélib' system in Paris are presented: this transit system offers more than $20'000$ bikes deployed in about $1'400$ stations that are in Paris and its border cities and twenty three trucks of capacity equal to 20 are used to move bikes during the day to match the demand. If the city is divided into areas on which only one vehicle operates, then each one would have to cover about 60 stations. The question that is here addressed is how to deal with a part of the city assigned to a single vehicle.

The problem can be formalized as follow. Let $G = (V, A)$ be a complete oriented graph where $V = \{0, \ldots, n\}$ is the vertex set composed by $n + 1$ vertices, the vertices in $\{1, \ldots, n\}$ representing the stations and the vertex 0 representing the depot and where A is the set of arcs. For each arc $(i, j) \in A$, we denote by c_{ij} the cost of the arc (i, j). The cost is assumed to satisfy the triangular inequality (i.e. $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$). Each vertex i has a capacity $C_i \in \mathbb{Z}_+$. For each vertex $i \in V$, its initial state in bikes is defined by $p_i \in \mathbb{Z}_+$ and its target, or final, state by $q_i \in \mathbb{Z}_+$. A vertex is *in excess* (resp. *in default*) if $p_i > q_i$

(resp. $p_i < q_i$). Some vertex can be *initially balanced* i.e. $p_i = q_i$. Moreover, throughout the chapter the *imbalance* $e_i = p_i - q_i$ is used and the depot is always assumed to have no bike: $C_0 = p_0 = q_0 = 0$ and $e_0 = 0$. The vehicle has also a capacity K.

A feasible solution for the SVOCPDP, also called a *route*, is a sequence of vertices, starting and finishing with the depot 0, together with bike displacements within the limits of capacity constraints, at the end of which the system is balanced: each vertex i has been brought from its initial state p_i to its target state q_i. In this case, the sequence of vertices is said to be *induced* by the route. The cost of the route is defined as the total travelled distance while following its sequence. The goal of the SVOCPDP is to find the minimal cost route. Note that the convergence for each vertex from p_i to q_i is not required to be monotonous: bikes can be loaded from vertices in default or unloaded at vertices in excess and transfers can take place at initially balanced vertices. Figure 2.1 shows an example of instance with 9 vertices (the depot and 8 stations) including one initially balanced vertex. A pair of values representing (p_i, q_i) is displayed next to each vertex and the capacity of the vehicle is equal to 8. Figure 2.2 shows a feasible solution.

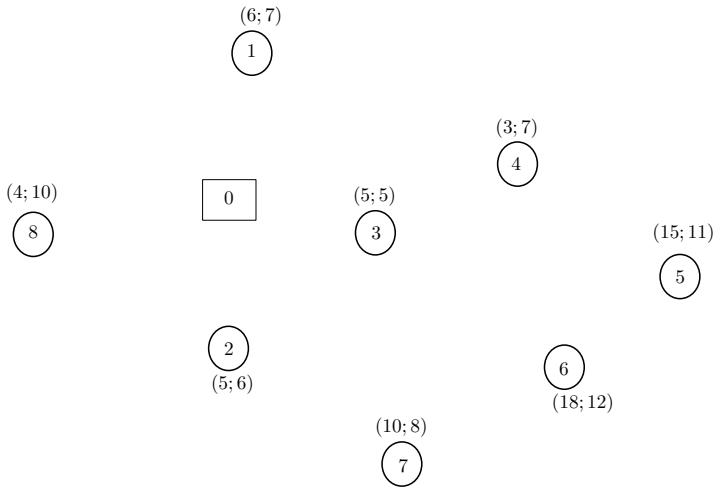

Figure 2.1: Example of an instance.

2.1.1 Complexity

The SVOCPDP is $\mathcal{NP}$-hard since it contains $\mathcal{NP}$-hard problems as special cases. It is easy to see it, but details are given for sake of completeness. The Travelling Salesman Problem (TSP) is obviously one of them. Set $p_i = 0$ and $q_i = 1$ for all vertices $i \in \{1, \ldots, n-1\}$, and

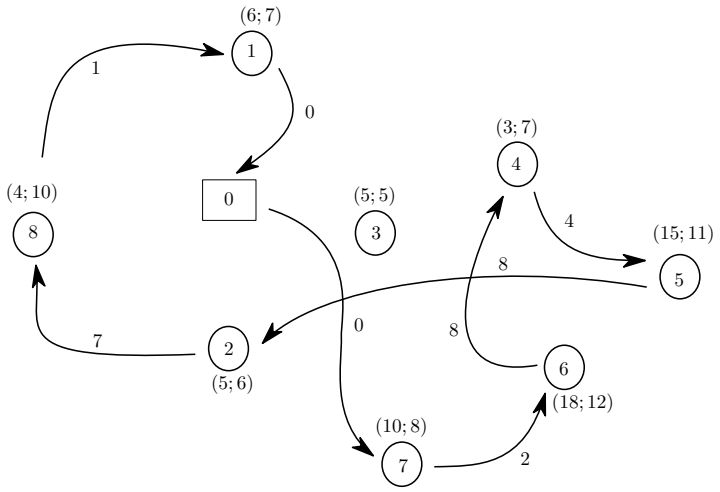

Figure 2.2: Example of a feasible solution (i.e. a route), when $Kgeq2$

set $p_n = n - 2$ and $q_n = 0$. Add a depot at distance 0 from the vertex n. Set $K = n - 2$. The optimal solution of the SVOCPDP coincides with the optimal TSP solution. The 2-partition problem is another special case. Let $b_1, \ldots, b_n$ be n non-negative integers (w.l.o.g. we assume that $\sum_i b_i$ is even). Define $m = \frac{1}{2}\sum_{i=1}^{n} b_i$. Take the complete graph with $n + 3$ vertices: n vertices with $p_i = b_i$ and $q_i = 0$; 2 vertices with $p_i = 0$ and $q_i = m$ and the depot. Define the c_{ij} to be 1 for each arc (i, j) and the capacity of the vehicle equal to m. The optimal solution of the SVOCPDP problem is equal to $n + 3$ if and only if there is a subset $I \subseteq \{b_1, \ldots, b_n\}$ such that $\sum_{i \in I} b_i = m$.

Moreover, an optimal route of the SVOCPDP problem may not have an encoding that is polynomial in the size of the input. The simple case with three vertices – the depot 0, one vertex 1 with $p_1 = B$ and $q_1 = 0$ and one vertex 2 with $p_2 = 0$ and $q_2 = B$ – is enlightening. Assuming that $K = 1$, the optimal sequence of vertices is $0, 1, 2, 1, 2, \ldots, 0$, with a number of terms $= 2B + 2$, although the input is in $O(\log_2 B)$.

2.1.2 Notations and basic notions

We define for all subsets $S \subseteq V$:

- $\bar{S} = V \backslash S$
- $\delta^+(S) := \{(i, j) \in A : i \in S; j \in \bar{S}\}$
- $\delta^-(S) := \{(i, j) \in A : i \in \bar{S}; j \in S\}$
- $\delta(0) := \delta^+(\{0\}) \cup \delta^-(\{0\})$

- $e(S) = \sum_{j \in S} e_j$
- $\mu(S)$ is equal to 1 whenever there is at least one initially non-balanced vertex in S, 0 otherwise.

Given $z \in \mathbb{R}_+^A$, $G[z]$ is the directed graph obtained from G by deleting the arcs a with $z_a = 0$. This graph is called the *support graph* of z.

A notion used several times in this chapter is the one of a b-*flow*. A b-flow is an usual notion if combinatorial optimization (see for instance [79, 55]). Given a directed graph $D = (U, A')$, a value $b \in \mathbb{R}^U$, and capacities $l, u \in \mathbb{R}^{A'}$ with $l \leq u$, a b-flow is a map $f : A' \to \mathbb{R}$ such that $l_a \leq f(a) \leq u_a$ for all $a \in A'$ and $\sum_{a \in \delta^+(v)} f(a) = b_v + \sum_{a \in \delta^-(v)} f(a)$ for all $v \in U$. If it exists, a b-flow can be computed in strongly polynomial time. Moreover, when all the b_v and the l_a, u_a are integral, if a b-flow exists, there is an integral one.

2.1.3 Plan

In Section 2.2, differences between the SVOCPDP and problems in the literature are outlined. Section 2.3 presents the proposition that enables to find in polynomial time the operations that bring the system to the possible state nearest to the target state, given a fixed sequence of vertices visited by the vehicle. An exact model of the problem is given in (Section 2.4), and a relaxation is presented in Section 2.5. In Section 2.6, we prove that deciding whether a solution of the relaxation problem is a feasible solution for the SVOCPDP is $\mathcal{NP}$-complete. The Section 2.7 contains the description of the branch-and-cut algorithm that is used to solve the relaxation. The proposition of Section 2.3 is used in Section 2.8 for deriving a tabu search. Finally, Section 2.9 presents the computational results on instances from the literature with a slight adaptation in order to fit the SVOCPDP's features.

2.2 Literature review

In the literature, similar problems are the One-commodity Pickup-and-Delivery Travelling Salesman Problem (1PDTSP) studied by Hernandez Pérez and Salazar González [67] and the swapping problem defined by Anily and Hassin [6]. This latter has been solved by [20, 47] in its preemptive and its non-preemptive versions. As for the 1PDTSP, several papers have been appeared recently in the literature which propose different exact and heuristic algorithms (see [68], [69] and [70]). Since these two problems present several similarities with our

problem, more details about them are given in the next paragraph. Raviv *et al.* [76] discuss different variants of the problem of repositioning the bikes which are modeled by mixed linear programs and solved using CPLEX. For some variants, they are able to solve several instances to optimality (up to 60 stations and 2 vehicles for their so-called "Arc-Indexed" variant). Our model is close to their "Sequence-Indexed" variant (see Section 3.4 of [76]), but instead of computing a minimum cost route with fixed target states, they try to find the best repartition of bikes that can be achieved by one or several vehicles within a time limit. Moreover, in the Sequence Index formulation given by [76], drops are not allowed, a thing that will be fixed in an upcoming version by [75].

The SVOCPDP gathers aspects from both the swapping problem and the 1PDTSP but differs by main features. In the swapping problem, a single vehicle has to move unitary objects from a vertex to another. There are m different types of objects and the vehicle has a unitary capacity: it can only move one object at the time. Shoshana Anily and Refael Hassin [6] showed that a vertex could be visited at most three times in the optimal solution. This theorem was the starting point of the work of Bordenave *et al.* [20, 47] for solving the swapping problem with a branch-and-cut algorithm. The SVOCPDP is different since there is only one type of object (bikes), but the supply and the demand are greater than one and the vehicle capacity is K. Therefore, Anily and Hassin's theorem does not hold anymore: for example in the trivial instance with one pickup vertex with $p_i = 5K$ and $q_i = 0$ and one delivery vertex with $p_i = 0$ and $q_i = 5K$, the vehicle has to do five round trips between the two vertices.

The main difference with the 1PDTSP is that in the SVOCPDP a vertex can be visited several times. The solution of the 1PDTSP is a feasible solution for the SVOCPDP, but the SVOCPDP can have a lower-cost optimal solution. Moreover, the 1PDTSP may have instances without feasible solutions (the instance of Figure 2.3, with $K = 3$, is an example), something that can not happen for the SVOCPDP.

The fact that one can get better solutions in routing problems when customers are allowed to be visited several times has already be noticed in other works. Note for instance the work by Archetti *et al.* [7], in which a split delivery problem is solved through a tabu search. In the problem we are dealing with any vertex can be used as a buffer where bikes can be indifferently temporary loaded or unloaded before being moved to their final destination. In particular, initially balanced vertices are not required to be visited, but may act as temporary depot in some optimal solutions.

The buffers can improve the optimal solution in some instances such as shown in the example given in Figure 2.3 and in Figure 2.5. In Figure 2.3 the square with the 0 inside is the depot and its distance with vertex 1 is null. For any other vertex i, $(p_i; q_i)$ is given. The capacity of the vehicle is $K = 3$ and the distances between all the peripheral vertices to the central one is 1. We draw here the optimal solution, whose value is equal to 10. In this solution a bike is temporary unloaded at the central vertex 6 and then reloaded on the vehicle. The optimal solution is equal to 12, if drop is forbidden because one of the odd number vertices would have to be visited twice.

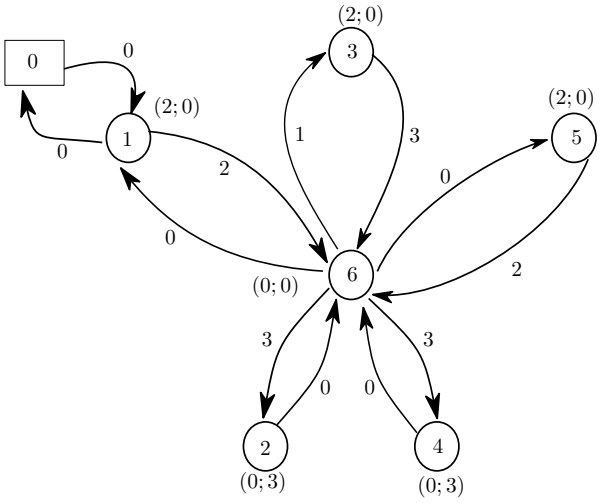

Figure 2.3: Star-framed network: example of how drops can help optimality

Figures 2.4 and 2.5 present a situation where the non-monotonous convergence of the load of the stations improves the solution. Figure 2.4 shows the best solution of the 1PDTSP. The two first vertices with initial and target states $(K; 0)$ and $(0; K)$ (K being the capacity of the vehicle) are here to prevent a use of the unlimited number of bikes in an optimal solution, available in the depot for the 1PDTSP. Figure 2.5 represents the optimal solution for the SVOCPDP. In both figures, the square with 0 is the depot, as in Figure 2.3. In Figure 2.5, the vehicle takes a bike from the left-corner vertex of the square, increasing momentary the deficit in bikes, before coming back with the two missing bikes. With Euclidean distance, the solution of the SVOCPDP is better than the one of the 1PDTSP.

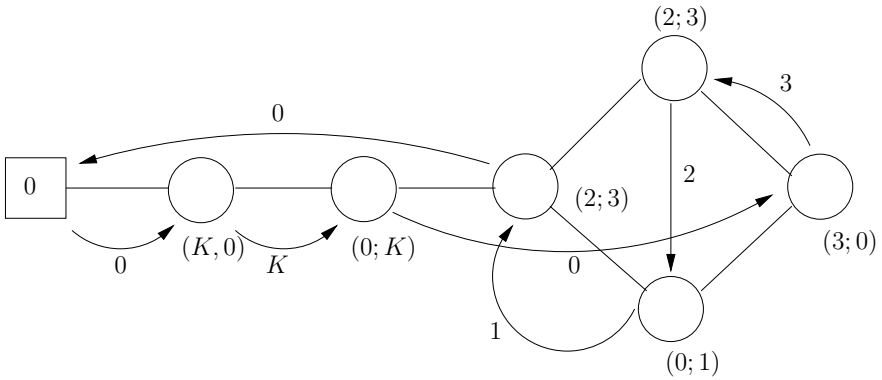

Figure 2.4: Square-framed network: an optimal solution for the 1PDTSP

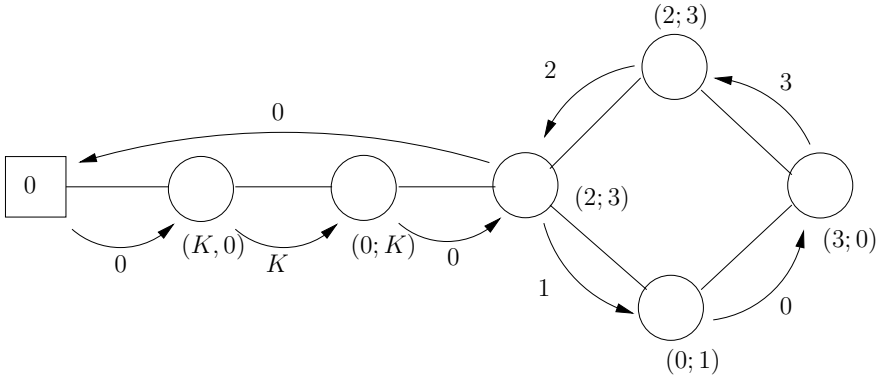

Figure 2.5: Square-framed network: an optimal solution for the SVOCPDP, the non-monotonous convergence helps

2.3 Dealing with sequences and routes

Recall that a route is a sequence of vertices, together with bike displacements within the limits of capacity constraints, at the end of which the system is balanced: each vertex i has been brought from its initial state p_i to its final state q_i. The difficulty of the SVOCPDP is that a feasible solution is identified by both a sequence of vertices and a set of numbers of bikes carried on arcs. The following proposition (and its companions Propositions 2.3.2 and 2.3.3) enables us to work only with sequences of vertices. It will be particularly useful in Section 2.8 when we will design a local search for the SVOCPDP.

Proposition 2.3.1. *Let $i_1, i_2 \ldots, i_k$ be a sequence of vertices, starting and finishing at the depot, $0 = i_1 = i_k$ being the depot. There is a polynomial algorithm finding new initial and target states (p'_i, q'_i) for each vertex i and a route inducing this sequence of vertices and satisfying these new states such that*

- *$0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i*
- *$\sum_i p'_i = \sum_i q'_i$*
- *the quantity $\sum_{i \in V} p'_i$ is maximal.*

In particular, it is possible to decide in polynomial time whether a sequence of vertices is induced by a route (in this later case, $p'_i = p_i$ for each vertex i).

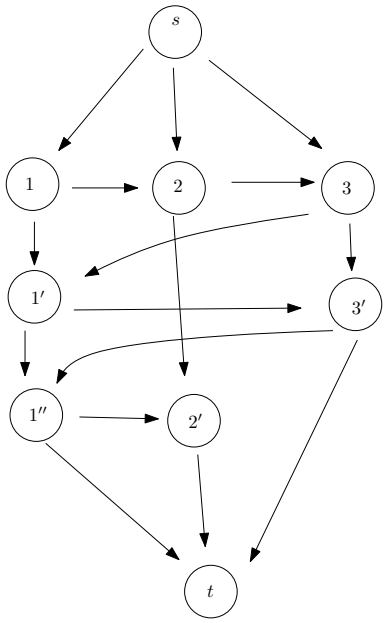

Figure 2.6: Example of the graph used in the algorithm of Proposition 2.3.1

The proposition says roughly speaking that it is possible to find the best bike displacements compatible with the sequence of vertices. The quantity $\sum_{i \in V}(q_i - q'_i)$ can be interpreted as a kind of "degree of infeasibility" of a sequence and the proposition shows how to minimize it polynomially.

Proof. Let us build an oriented graph $D = (U, A')$ as follows. U has $k + 2$ elements: each vertex i_j (make as many copies of a vertex i of G as there are occurrences of i in the sequence) and two more vertices s and t. The arcs in A' are of four types:

1. one arc between s and the first occurrence of each vertex i in the sequence, with capacity p_i,
2. one arc (i_j, i_{j+1}) for each $j = 1, \ldots, k-1$, with capacity K,
3. one arc between the rth occurrence of each vertex i and its $(r+1)$th occurrence, if there is one, with capacity C_i,
4. one arc between the last occurrence of each vertex i in the sequence and t, with capacity q_i.

See Figure 2.6 for an illustration with the sequence $0 \to 1 \to 2 \to 3 \to 1 \to 3 \to 1 \to 2 \to 0$. Computing a maximum s-t flow in this graph leads to the proposition. Indeed, any s-t flow on D encodes possible bike displacements compatible with the given sequence of vertices. The numbers of bikes to be moved while going from i_j to i_{j+1} are given by the flow on arc (i_j, i_{j+1}) (arcs defined in 2.); the number of bikes remaining in a vertex i after the rth visit of the vehicle is given by the flow on arc between the rth occurrence of vertex i and its $(r+1)$th occurrence (arcs defined in 3.); the initial and final numbers of bikes in a vertex i are given respectively by the flows on arcs defined in 1. and 4. And conversely, any bike displacements compatible with the given sequence of vertices induce an s-t flow.

p'_i is then the value of the flow in the arc between s and the first occurrence of i, and q'_i the value of the flow in the arc between the last occurrence of i and t. If a vertex i is not present in the sequence, then we set $p'_i = q'_i = \min(p_i, q_i)$. □

In the case that all C_i are sufficiently large, Proposition 2.3.1 has a nice and maybe quite unexpected corollary, formalized by the following proposition. If we are not allowed to change the initial states, but only the final ones, the solution given by Proposition 2.3.1 is a route with new final states closest to the original ones.

Proposition 2.3.2. *Assume that we have $C_i = +\infty$ for all vertices i. Let $0 = i_1, i_2 \ldots, i_k = 0$ be a sequence of vertices. Consider the problem of finding bike displacements along this sequence leading to the final state $\tilde{q}$ closest to q when starting from the initial state p, where "closest" means the state that minimizes the L_1 norm $||\tilde{q} - q||_1 = \sum_{i \in V} |\tilde{q}_i - q_i|$.*

The bike displacements given by Proposition 2.3.1 for this sequence are precisely the solution of this problem.

Proof. Denote by $\mathcal{Q}$ the set of all target states $\tilde{q}_i$ such that there exists a route for initial states p_i and target states $\tilde{q}_i$ inducing the sequence. We are interested in the solution of $\min_{\tilde{q} \in \mathcal{Q}} ||\tilde{q} - q||_1$.

We first show that for any solution of $\min_{\tilde{q} \in \mathcal{Q}} ||\tilde{q} - q||_1$, there is a route for new initial and target states p'_i and q'_i, with

- $0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i
- $\sum_i p'_i = \sum_i q'_i$

and such that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

We have the following central observation: let $\tilde{q}' \in \mathcal{Q}$; there exists $\tilde{q} \in \mathcal{Q}$ such that $||\tilde{q} - q||_1 \leq ||\tilde{q}' - q||_1$ and such that, on each vertex i, there are at least $\max(0, \tilde{q}_i - q_i)$ bikes that have not been moved. Indeed, for each vertex i, choose $\max(0, \tilde{q}'_i - q_i)$ bikes among the $\tilde{q}'_i$ bikes at the end of the route. Consider now the solution $\tilde{q}$ obtained with the same bike displacements except for these bikes, which are not allowed to leave their initial vertices. Denoting by $\bar{p}_i$ the number of bikes on vertex i that are not allowed to move, we have

$$\sum_i \bar{p}_i = \sum_{i:\, q_i < \tilde{q}'_i} \tilde{q}'_i - q_i$$

and

$$\tilde{q}_i = \begin{cases} \tilde{q}'_i + \bar{p}_i & \text{for each vertex } i \text{ such that } q_i \geq \tilde{q}'_i \\ q_i + \bar{p}_i & \text{for each vertex } i \text{ such that } q_i < \tilde{q}'_i \end{cases}$$

Note that $\bar{p}_i \geq \tilde{q}_i - q_i$. The distance to q is equal to

$$||\tilde{q} - q||_1 = \sum_{i:\, q_i \geq \tilde{q}'_i} |\tilde{q}_i - q_i| + \sum_{i:\, q_i < \tilde{q}'_i} |\tilde{q}_i - q_i| \leq \sum_{i:\, q_i \geq \tilde{q}'_i} |\tilde{q}'_i - q_i| + \sum_{i:\, q_i \geq \tilde{q}'_i} \bar{p}_i + \sum_{i:\, q_i < \tilde{q}'_i} \bar{p}_i = \sum_i |\tilde{q}'_i - q_i| = ||\tilde{q}' - q||_1.$$

We can therefore choose the target states $\tilde{q}$ minimizing $||\tilde{q} - q||_1$ such that at least $\max(0, \tilde{q}_i - q_i)$ bikes have not been moved for each vertex i. Let us now define, for the route with these target states, p_i^M the number of bikes that have left vertex i and q_i^M the number of bikes that have been brought to vertex i. Note that according to our choice for $\tilde{q}$, we have

$q_i^M \leq q_i$ (and of course $p_i^M \leq p_i$). We have

$$||\tilde{q} - q||_1 = \sum_{i \in V} |\tilde{q}_i - q_i| = \sum_{i \in V} |p_i - p_i^M + q_i^M - q_i|. \tag{2.1}$$

Let $\delta_i := \min(p_i - p_i^M, q_i - q_i^M)$ and define $p'_i := p_i^M + \delta_i$ and $q'_i := q_i^M + \delta_i$ for each vertex i. Note that the initial and target states p'_i and q'_i are feasible in the sense of Proposition 2.3.1. According to Equation (2.1), we have $||\tilde{q} - q||_1 = \sum_{i \in V} |p_i - p'_i + q'_i - q_i|$. Using the fact that for each i, at least one of $p_i - p'_i$ and $q_i - q'_i$ is equal to 0, we get that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

Conversely, assume that we have a route for initial and target states p'_i and q'_i, with

- $0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i
- $\sum_i p'_i = \sum_i q'_i$
- the quantity $\sum_{i \in V} p'_i$ is maximal.

We now show that there is a solution of $\min_{\tilde{q} \in \mathcal{Q}} ||\tilde{q} - q||_1$ such that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

Since $C = +\infty$, if $p'_i < p_i$ and $q'_i < q_i$, we could have taken into account one more bike on i (increasing by one p'_i), which would have not left vertex i. Therefore, for each i, we have at least one of $p_i - p'_i$ and $q_i - q'_i$ that is equal to 0 and thus, we have $2 \sum_{i \in V} (p_i - p'_i) = \sum_{i \in V} |p_i - p'_i + q'_i - q_i|$. Let us now consider what we have as a final state when we start with p_i and make the same bike displacements as in the route: we get $\tilde{q}_i := p_i - p'_i + q'_i$ bikes on each vertex i, for which we have $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$ as required. □

As a by-product of the proof of Proposition 2.3.1, we get

Proposition 2.3.3. *Assume that we have a feasible solution to the SVOCPDP with fractional numbers of bikes carried during some moves. Then there is also a feasible solution of the same cost with only integral numbers of bikes carried during the moves.*

Thanks to this proposition, we can "forget" the integrality constraint without changing the cost of the optimal solution.

2.4 An exact model

The exact model is based on the assumption that for each vertex i is given a constant β_i, which is an upper bound of the number of times the vehicle has to visit i in any optimal solution.

For instance, given an upper bound (i.e. a feasible solution) with total cost O, we can set $\beta_i := O/\min_{j\neq i} c_{ij}$ ([60]). The linear relaxation of this model could be very weak and therefore useless, and moreover the model can be intractable, since β_i computed in this way can be quite huge and the model involves variables with four indices. It would be interesting to compute a tighter β_i, which would reduce the size of the exact model, but whether it is possible from *a priori* considerations remains an open question.

We introduce the following variables. $x_{i,t}$ takes the value 1 only if vertex i is visited at least t times. $z_{i,t,i',t'}$ takes the value 1 only if the vehicle visits vertex i' for the t'th time just after having visited i for the tth time, and $y_{i,t,i',t'}$ is then the number of bikes that is carried by the vehicle during this move. The variables y are not required to be integral since, according to Proposition 2.3.3, it does not change the optimal value of the linear program. $u_{i,t,i',t'}$ is a variable which takes the value 1 if the t'th visit of vertex i' comes in the route after the tth time of vertex i (but not necessarily right after). Then we can write the model as follows:

$$(P)\quad z(P) = \min \sum_{(i,i')\in A} \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} c_{(i,i')} z_{i,t,i',t'}$$

s.t.

$$x_{i,t} \leq x_{i,t-1} \qquad \forall i \in V,\ \forall t \in \{2,\dots,\beta_i\} \tag{2.2}$$

$$\sum_{i'\in V\setminus\{i\}} \sum_{t'=1}^{\beta_{i'}} z_{i,t,i',t'} = x_{i,t} \qquad \forall i \in V,\ \forall t \in \{1,\dots,\beta_i\} \tag{2.3}$$

$$\sum_{i'\in V\setminus\{i\}} \sum_{t'=1}^{\beta_{i'}} z_{i',t',i,t} = x_{i,t} \qquad \forall i \in V,\ \forall t \in \{1,\dots,\beta_i\} \tag{2.4}$$

$$\sum_{i\in V\setminus\{0\}} \sum_{t=1}^{\beta_i} z_{0,1,i,t} = 1 \tag{2.5}$$

$$u_{i,t,i',t'} \geq u_{i,t,i'',t''} + z_{i'',t'',i',t'} - 1 \qquad \forall i,i',i'' \in V,\ \forall t \in \{1,\dots,\beta_i\},\ \forall t' \in \{1,\dots,\beta_{i'}\},\ \forall t'' \in \{1,\dots,\beta_{i''}\} \tag{2.6}$$

$$u_{i,t,i',t'} \geq z_{i,t,i',t'} \qquad \forall i,i' \in V, \forall t \in \{1,\dots,\beta_i\},\ \forall t' \in \{1,\dots,\beta_{i'}\} \tag{2.7}$$

$$u_{i,t,i,\tau} = 0 \qquad \forall i \in V, \forall t,\tau \in \{1,\dots,\beta_i\} \text{ with } \tau \leq t \tag{2.8}$$

$$\sum_{i\in V\setminus\{0\}} \sum_{t=1}^{\beta_i} y_{0,1,i,t} = 0 \tag{2.9}$$

$$y_{i,t,i',t'} \leq K z_{i,t,i',t'} \qquad \forall i, i' \in V, \forall t \in \{1, \ldots, \beta_i\}, \forall t' \in \{1, \ldots, \beta_{i'}\} \tag{2.10}$$

$$0 \leq \sum_{i' \in V \setminus \{i\}} \sum_{\tau=1}^{t} \sum_{\tau'=1}^{\beta_{i'}} (y_{i',\tau',i,t} - y_{i,t,i',\tau'}) + p_i \leq C_i \quad \forall i \in V, \forall t \in \{1, \ldots, \beta_i\} \tag{2.11}$$

$$\sum_{i' \in V \setminus \{i\}} \sum_{\tau=1}^{\beta_i} \sum_{\tau'=1}^{\beta_{i'}} (y_{i',\tau',i,t} - y_{i,t,i',\tau'}) + p_i = q_i \qquad \forall i \in V \tag{2.12}$$

$$z_{i,t,i',t'}, u_{i,t,i',t'} \in \{0,1\} \qquad \forall i, i' \in V, \forall t \in \{1, \ldots, \beta_i\}, \forall t' \in \{1, \ldots, \beta_{i'}\} \tag{2.13}$$

$$y_{i,t,i',t'} \in \mathbb{R}_+ \qquad \forall i, i' \in V, \forall t \in \{1, \ldots, \beta_i\}, \forall t' \in \{1, \ldots, \beta_{i'}\} \tag{2.14}$$

$$x_{i,t} \in \{0,1\} \qquad \forall i \in V, \forall t \in \{1, \ldots, \beta_i\} \tag{2.15}$$

A solution of the SVOCPDP can be seen as an elementary circuit in the directed graph $\mathcal{D}$ whose vertices are the (i,t) with $i \in V$ and $t \in \{1, \ldots, \beta_i\}$ and the arcs are the $((i,t),(i',t'))$. Going through (i,t) means for the SVOCPDP going through i for the tth time.

Constraints (2.2) are logical constraints implied by the definition of $x_{i,t}$. Constraints (2.3), (2.4) and (2.5) and and (2.9) ensure that we get in $\mathcal{D}$ a collection of vertex-disjoint circuits, at least one of which going through $(0,1)$ (the vehicle leaves the depot at least once with an empty load). Constraints (2.6), (2.7) and (2.8) ensure that the circuit in $\mathcal{D}$ is coherent with the t index: the tth visit of vertex i comes after the τth visit, for any $\tau < t$. Note that we get for free that we have only one circuit in $\mathcal{D}$: constraints (2.6) imply that the t visit of i must be strictly after the tth visit of i, a contradiction.

Constraints (2.10) limit the number of bikes transshipped on an arc at each move to the capacity of the vehicle. Constraints (2.11) bound the number of bikes parked at a vertex i at any time step between 0 and its maximal capacity C_i. Constraints (2.13), (2.14) and (2.15) ensure that the vertices have all reached their target endowment at the end of the route.

Remark

The coherence on the t indices implied by the variables $u_{i,t,i',t'}$ and the constraints (2.6), (2.7) and (2.8) in (P) can alternatively be obtained by the introduction of variables $h_{i,t} \in \{1, \ldots, \sum_{i=1}^{n} \beta_i\}$ encoding the instant of tth passage on vertex i and the use of "big-M" constraints, in a similar spirit as for the TSP with time windows, see for instance [33].

2.5 Relaxations

This section presents two equivalent mixed integer linear programming problems. They represent a relaxation of the original problem, since even solved to optimality they produce a lower bound of the optimal solution of the problem.

2.5.1 A first relaxation

Using the variables of the exact model (P) of Section 2.4, a relaxation for the SVOCPDP can be obtained by defining $z_{(i,i')} = \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} z_{i,t,i',t'}$, which represents the number of times the arc $(i,j) \in A$ is traversed in the solution, and by defining $y_{(i,i')} = \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} y_{i,t,i',t'}$, which represents the total number of bikes transported on it.

The relaxation is:

$$(RP1) \quad z(RP1) = \min \sum_{(i,j)\in A} c_{ij} z_{ij} \tag{2.16}$$

s.t.

$$\sum_{j\in V} z_{ij} = \sum_{j\in V} z_{ji} \qquad \forall i \in V \tag{2.17}$$

$$\sum_{i\in V\setminus\{0\}} z_{0i} = 1 \tag{2.18}$$

$$\sum_{i\in V\setminus\{0\}} y_{0i} = 0 \tag{2.19}$$

$$p_i + \sum_{j\in V\setminus\{i\}} y_{ji} = q_i + \sum_{j\in V\setminus\{i\}} y_{ij} \qquad \forall i \in V \tag{2.20}$$

$$\sum_{(i,j)\in\delta^+(S)} z_{ij} \geq \mu(S) \qquad S \subseteq V \setminus \{0\} \tag{2.21}$$

$$0 \leq y_{ij} \leq K z_{ij} \qquad \forall (i,j) \in A \tag{2.22}$$

$$z_{ij} \in \mathbb{Z}_+, \qquad \forall (i,j) \in A \tag{2.23}$$

Again, requiring that the y_{ij} are integral does not improve the value of the relaxation. Indeed, whenever the values of the z_{ij} are fixed, the variables y_{ij} are solutions of a b-flow problem, and hence, since the p_i, q_i and K are integer numbers, there is an optimal solution with integral values for the y_{ij}.

Constraints (2.17) and (2.18) come from the constraints (2.3), (2.4) and (2.5). Constraints (2.19), (2.20) and (2.22) come from the constraints (2.10) and (2.11). Finally, constraints (2.21) ensure the connectivity of the solution, while initially balanced vertices might be skipped in an optimal solution.

The proposed relaxation does not take into account the evolution on the number of bikes on each vertex at each time-step: all the moves are considered simultaneously and so the sequential dimension of the problem disappears. Therefore, it is worth nothing saying that a solution of the original problem SVOCPDP provides a solution of (RP1). Nevertheless, a solution of model (RP1) might not be a solution of the SVOCPDP, as shown by the example given in Figure 2.7. In this example the values $(p_i; q_i)$ are displayed next to each vertex, $K \geq 2$ and the optimal solution of (RP1) is represented: each arc a in the figure corresponds to $z_a = 1$, the others z_a being equal to 0 and the numbers near each arc are the y_a. The optimal solution satisfies model (RP1), but violates SVOCPDP: the vehicle would have to take one bike from the first vertex that is not yet arrived there.

However, the solution of the previous linear program is often a solution to the SVOCPDP as illustrated in the computational results section (Section 2.9).

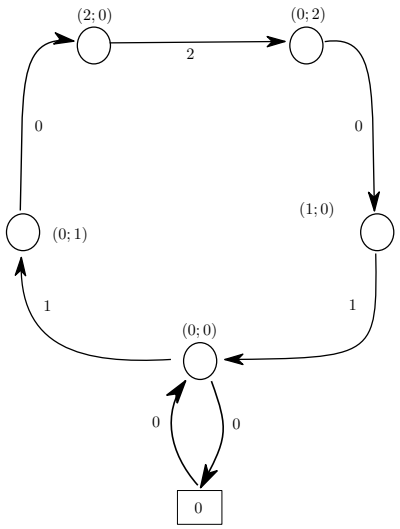

Figure 2.7: A solution of (RP1) that is not a solution of the SVOCPDP

2.5.2 A second relaxation

The former integer linear program requires two families of variables z_{ij} and y_{ij}. Linear program solvers are sensitive to the number of variables, and even if the y_{ij} can be assumed to be real, (RP1) is too big to be solved by standard solvers. A way to get a tractable formulation consists in reducing the number of variables. Therefore, we now define a new linear program which contains only the z_{ij} variables and we show that solving the new integer linear program model is equivalent to solving the former one.

$$(RP2) \quad z(RP2) = \min \sum_{(i,j)\in A} c_{ij} z_{ij} \tag{2.24}$$

s.t.

$$\sum_{j\in V} z_{ij} = \sum_{j\in V} z_{ji} \qquad \forall i \in V \tag{2.25}$$

$$\sum_{i\in V\setminus\{0\}} z_{0i} = 1 \tag{2.26}$$

$$\sum_{(i,j)\in\delta^+(S)} z_{ij} \geq \mu(S), \qquad \forall S \subseteq V \setminus \{0\} \tag{2.27}$$

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{ij} \geq \left\lceil \frac{e(S)}{K} \right\rceil \qquad \forall S \subseteq V \tag{2.28}$$

$$z_{ij} \in \mathbb{Z}_+, \qquad \forall (i,j) \in A \tag{2.29}$$

Any feasible solution of the SVOCPDP induces z_{ij} that satisfy the constraints of program (RP2). Constraints (2.25), (2.26) and (2.27) are similar to those of the first relaxation (RP1). Constraints (2.28) are the capacity constraints saying that, for any subset $S \subseteq V$, the vehicle must goes at least $\left\lceil \frac{|e(S)|}{K} \right\rceil$ times into S. The absolute value $|\cdot|$ is useless in (RP2). Indeed, if $e(S) \geq 0$, it is not necessary. If $e(S) < 0$, (2.28) are redundant and in anyway, when (2.28) is written with $\bar{S}$, we obtain precisely what would have been obtained with the $|.|$ for S.

Constraints (2.28) are well-known in the Capacitated Vehicle Routing Problem (CVRP) context and it is still an open question if a polynomial algorithm separating these constraints exists (see for example [64]). In Subsection (2.7.2), we explain how to deal with them.

Variables y_{ij} disappear in (RP2). However, for any feasible solution z_{ij} of (RP2), there is a feasible solution to (RP1) with the same z_{ij} (and hence the same value of the objective function), and conversely. This fact is summarized in the following proposition.

Proposition 2.5.1. *Let y, z be a feasible solution of (RP1). Then z is a feasible solution of (RP2). Conversely, let z be a solution of (RP2). Then there exists y such that y, z is a feasible solution of (RP1).*

This property is also proven by [66] when z encodes an Hamiltonian circuit. Here the proposition is proven in a more general case, and the proof is maybe slightly simpler since it does not involve Bender's decomposition but the cut condition for b-flows.

Proof. Take a feasible solution y, z of (RP1). We show that the z_{ij} satisfy (RP2). The only thing that has to be checked is that the z_{ij} satisfy Constraints (2.28). Let $S \subseteq V$. Using Constraints (2.19) and (2.22), aggregated over $(i, j) \in S$ we get

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{ij} \geq \frac{1}{K} \sum_{(i,j)\in\delta^+(S)} y_{ij}.$$

Hence

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{ij} \geq \frac{1}{K} \left(\sum_{(i,j)\in\delta^+(S)} y_{ij} - \sum_{(i,j)\in\delta^-(S)} y_{ij} \right).$$

Since

$$\sum_{(i,j)\in\delta^+(S)} y_{ij} - \sum_{(i,j)\in\delta^-(S)} y_{ij} = \sum_{i\in S} \left(\sum_{j\in V\setminus\{i\}} y_{ij} - \sum_{j\in V\setminus\{i\}} y_{(j,i)} \right),$$

we get with the help of constraints (2.20) that

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{ij} \geq \frac{1}{K} e(S).$$

Conversely, take a feasible solution z of (RP2). The only thing that has to be checked is that there exists a non-negative b-flow y with $b(i) = p_i - q_i$ for all $i \in V$ with a capacity equal to $K z_{ij}$ on each arc (i, j). But constraints (2.28) are precisely the well-known *cut condition* for flows on networks (see for instance [45]). The integrality is a consequence of the integrality of $b(i)$. □

Note that the two programs (RP1) and (RP2) are $\mathcal{NP}$-hard, since they obviously contain the TSP as a special case. Moreover, the continuous relaxation of (RP2) provides a better lower bound than the continuous relaxation of (RP1) thanks to $\lceil \cdot \rceil$ in Constraints (2.28).

2.6 Relaxation vs original problem

The hardness of SVOCPDP, already emphasized in Subsection 2.1.1, appears also through the following four propositions, which show that even if we have a feasible solution or an optimal solution of (RP1) or (RP2), it is an $\mathcal{NP}$-complete problem to decide whether this solution is induced by a feasible solution for the SVOCPDP.

Proposition 2.6.1. *Let y, z be a feasible solution of (RP1). Deciding whether there is a feasible solution of the* SVOCPDP *inducing y, z is $\mathcal{NP}$-complete.*

Proof. Let $b_1, \ldots, b_r$ be r non-negative integers (w.l.o.g. we assume that $\sum_l b_l$ is even). Define $m = \frac{1}{2}\sum_{l=1}^{r} b_l$. Consider then the graph of Figure 2.8. It encodes a feasible solution y, z of program (RP1): each arc (i, j) has $z_{ij} = 1$ and each number next to it is the corresponding value for y_{ij}. Assume that the capacity of the vehicle is $m + 1$. We are going to prove now that there is a route inducing such y_{ij}, z_{ij} if and only if there is a subset $I \subseteq \{1, \ldots, r\}$ such that $\sum_{l\in I} b_l = m$, whence showing that deciding whether such a route exists is $\mathcal{NP}$-complete. Note that there are exactly $2m + 1$ bikes in the network.

Assume that such a route exists. Consider again Figure 2.8 and the following instant: the vehicle takes arc (u, u'). At this time, the vehicle carries $m + 1$ bikes. Therefore, it has already taken the arc (s, u) (in order to have m bikes) and exactly one of the arcs (v, v') (in order to get the remaining bike). It cannot have already taken the other arc (v, v'), otherwise it should go back to the depot. It has not taken the arc (u', s) yet. Thus, $m + 1$ bikes among the $2m + 1$ are on the vehicle, and m bikes are on vertex v': there is no bike left on vertex s. When the vehicle goes back to vertex s, using arc (u', s), it carries exactly m bikes. These m bikes have to be carried to vertex v, using arcs (s, v). Hence, the algorithm has to identify a subset $I \subseteq \{1, \ldots, r\}$ such that $\sum_{l\in I} b_l = m$.

Conversely, assume that a subset I such that $\sum_{l\in I} b_l = m$ has been identified. The sequence starting with the depot, then going through s, u, s in this order, then going back and forth between s and v, using the arcs indexed by I, to carry m bikes to v, then going through v', u, u', s in this order, then using the remaining arcs between s and v, and finally going through v, v' in order to finish again at the depot is a route. □

Proposition 2.6.2. *Let z be a feasible solution of (RP2). Deciding whether there is a feasible solution of the* SVOCPDP *inducing z is $\mathcal{NP}$-complete.*

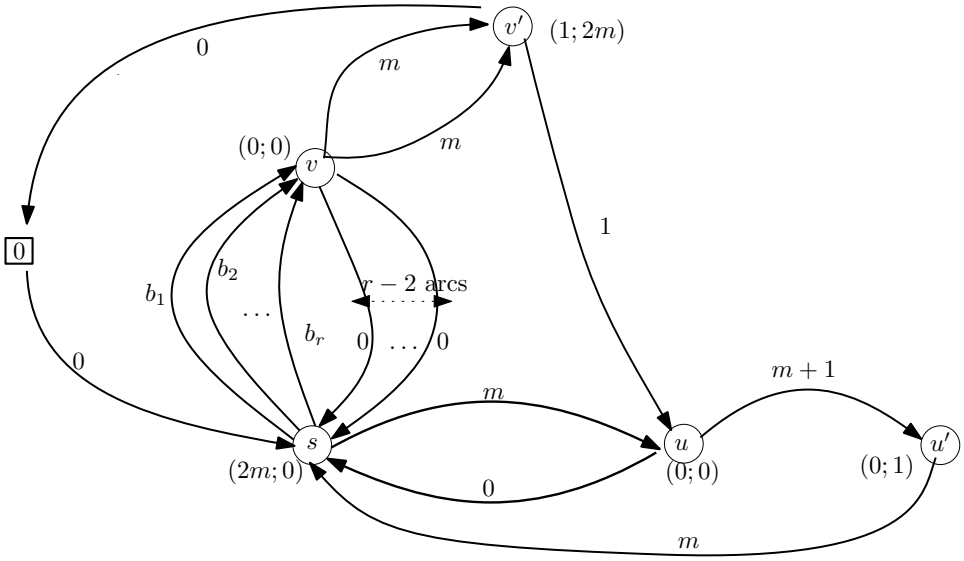

Figure 2.8: Proof of $\mathcal{NP}$-completeness

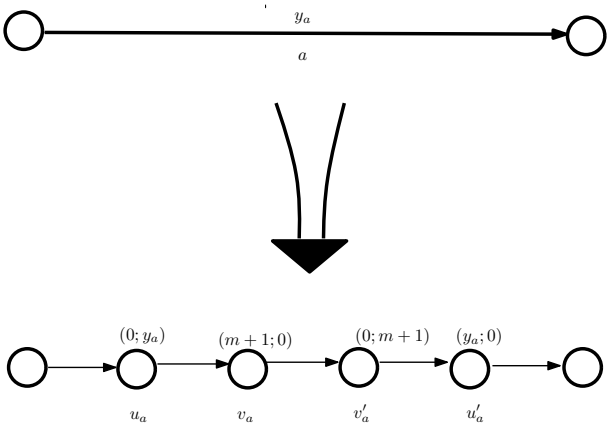

Figure 2.9: Construction of G' in the proofs of Propositions 2.6.2, 2.6.3, 2.6.4

Proof. The proof is very similar to the one of Proposition 2.6.1. Again, we work with a reduction from the 2-partition problem and with the graph of Figure 2.8. In order to adapt the proof for (RP2), we simulate the y_{ij} by subdividing each arc $a = (i, j)$ in five arcs, introducing four new vertices u_a, u'_a, v_a and v'_a, with $p_{u_a} = 0$, $q_{u_a} = y_{ij}$, $p_{v_a} = m + 1$, $q_{v_a} = 0$, $p_{v'_a} = 0$, $q_{v'_a} = m + 1$, $p_{u'_a} = y_{ij}$, $q_{u'_a} = 0$. Figure 2.9 illustrates this transformation. The capacity of the vehicle is set to $m + 1$. The vertices of this new graph G' provide an instance of the SVOCPDP, and the arcs encode a feasible solution of (RP2) for this instance. The solution satisfies constraints (2.28) since there are values y_{ij} satisfying constraints (2.20).

Because of the construction, these values y_{ij} are unique and entirely determined by the initial and target states on each vertex. If there is a route on the new graph, it will induce a route on the original graph of Figure 2.8. This route will imply the existence of a 2-partition, for the same reasons as those exposed in the proof of Proposition 2.6.1. Conversely, if there is a 2-partition, it implies a route for the original graph, which will in turn implies a route for the new graph. □

Those two propositions deal with feasible solutions of programs (RP1) and (RP2). Now, even if we have a special solution, especially the optimal one, the question whether it is induced by an optimal one of the SVOCPDP is $\mathcal{NP}$-complete. We have indeed the following propositions.

Proposition 2.6.3. *Let z^* be an optimal solution of (RP2). Deciding whether there is a feasible solution of the* SVOCPDP *inducing z^* is $\mathcal{NP}$-complete.*

Proof. We can force the solution built in the proof of Proposition 2.6.2 to be optimal, and then the same proof does the job. To force the solution to be optimal, we consider now the graph G' obtained in the proof of Proposition 2.6.2 by subdividing each arc in five new arcs. We assume that the cost to traverse each of these arcs is 1. Now, we build from this graph a complete graph for which each arc (i, j) gets for cost the cost of a shortest path between i and j. This complete graph provides an instance of the SVOCPDP. The arc of G' encodes a solution z for (RP2), which is optimal since each new vertex u_a, u'_a, v_a, v'_a has to be visited. □

Proposition 2.6.4. *Let y^*, z^* be an optimal solution of (RP1). Deciding whether there is a feasible solution of the* SVOCPDP *inducing y^*, z^* is $\mathcal{NP}$-complete.*

Proof. There is only one solution for y once z has been defined according to the proof of Proposition 2.6.2, whence the same proof as for Proposition 2.6.3 works. □

2.7 Lower bound

The integer linear program (RP2) has an exponential number of capacity and connectivity constraints. The problem is solved through a branch-and-cut algorithm. At each node of the branch-and-cut tree the continuous relaxation of the problem (RP2) is solved, but only a subset of constraints (2.27) and (2.28) are activated. The continuous optimal solution $\bar{z}$ is checked by different routines that try to determinate violated constraints. If a violated constraint is found, it is added into the linear relaxation which is again optimally solved. If no violated constraint is found, but $\bar{z}$ is still fractional, branching starts.

If the solution is integer, then it is a feasible solution of (RP2). Its value is kept and can be used as an upper bound on the optimal value of (RP2). When the algorithm reaches the end, updating the upper bound during the branch-and-cut process gives at the end the optimal solution of (RP2). If it stops before (for instance if there is a time limit), we get at the end the best feasible solution encountered during its exploration and a lower bound on the optimal solution value, used for calculating the gap.

2.7.1 Separation of connectivity constraints

Given a fractional solution $\bar{z}$, computed at a node of the branch-and-cut tree, we check that constraints (2.27) are satisfied as follows. We consider the support graph $G[\bar{z}]$ together with a capacity equal to $\bar{z}_{ij}$ on each arc (i, j). For each unbalanced vertex i, we compute the minimum cut $\delta^+(S)$ separating i from the depot 0 (with the algorithm by [38]). If this minimum cut has a value strictly lower than 1, then we add the corresponding constraint to the linear program.

2.7.2 Separation of capacity constraints

As already noted when we have defined (RP2), there is no known polynomial algorithm that checks the capacity constraints (2.28).

The following constraints are less tight but can be separated in polynomial time.

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{ij} \geq \frac{e(S)}{K}, \quad (\forall S \subseteq V) \tag{2.30}$$

These constraints are the relaxation of the constraints (2.28) in (RP2). They are called "relaxed capacity constraints". A method for finding violated relaxed capacity constraints for

the CVRP is proposed by [64]. In the SVOCPDP the difference is that a vertex can either be in excess or in default. Therefore their method has been adapted to fit the specific characteristics of the SVOCPDP. The two arcs connecting the vertex 0 with the other vertices are deleted and two new vertices s and t are added. Vertex s is linked to all vertices in excess with the capacity $\kappa_{si} = \frac{e_i}{K}$ for each arc whereas all vertices in default are linked to vertex t with the capacity $\kappa_{it} = \frac{-e_i}{K}$. The capacity is equal to $\bar{z}_{ij}$ on the original arcs (i, j).

We compute an s-t min-cut (again with the Edmonds-Karp algorithm [38]). Let X be the subset of vertices such that $\delta^+(X)$ is the s-t min-cut. s is in X and we define $S := X \setminus \{s\}$. We still define $\bar{S}$ as $V \setminus S$, where V is the vertex set of the original graph, and contains neither s nor t.

$$\text{capacity of } \delta^+(X) = \sum_{i \in S \setminus \{0\}, j \in \bar{S} \setminus \{0\}} \bar{z}_{ij} + \sum_{i \in \bar{S} \setminus \{0\}} \kappa_{si} + \sum_{i \in S \setminus \{0\}} \kappa_{it} \tag{2.31}$$

$$= \sum_{(i,j) \in \delta^+(S) \setminus \delta(0)} \bar{z}_{ij} - \frac{e(S)}{K} + \sum_{i \in V \colon e_i > 0} \frac{e_i}{K}. \tag{2.32}$$

If the capacity of $\delta^+(X)$ minus $\sum_{i \in V \colon e_i > 0} \frac{e_i}{K}$ (which is a fixed value) is less than 0, it means that a relaxed capacity constraint is violated and the corresponding constraint are added to the linear relaxation. Otherwise, there is no violated relaxed capacity constraint as shown in Figure 2.10, which presents a possible solution of the linear relaxation when solving the instance presented in Figure 2.2, but the capacity of the vehicle is $K = 10$. The fractional value of each arc is given next to it.

When relaxed capacity constraints are respected a second procedure runs looking for violated capacity constraints. The interest to look for these constraints in addition to the former is to increase the speed of the general algorithm. Indeed, we get in this case tighter constraints. The used procedure is again an heuristic presented in [64], since there is no known polynomial algorithm. It is a tabu search that tries to find a subset S violating the constraints. A short explanation of the tabu search method is done later in Section 2.8. The main idea is starting from a subset S of vertices, vertices are either added to S or removed from S in order to find a new subset where capacity constraints are violated. Roughly speaking, the vertex that is added to or removed from the subset is kept in a tabu list and cannot be used for a given number of iterations. If the procedure finds a violated capacity constraint, it is added to the linear program.

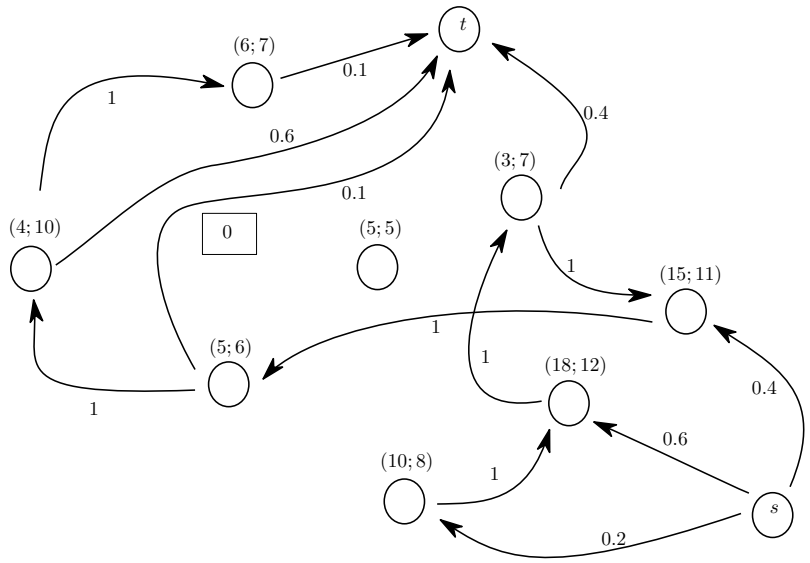

Figure 2.10: Construction of the new graph from the graph of Figure 2.2 to check the relaxed capacity constraints

2.7.3 Initial relaxation, separation strategy and branching rules

The initial relaxation solved is (RP2) without (2.27) and with (2.28) only for $S = \{i\}$ for all $i \in V$. The separation routines are called with respect to the following order:

- connectivity constraints (2.27)
- relaxed capacity constraints (2.30)
- capacity constraints (2.28)

The branch-and-bound tree framework SCIP has been used for the computational results. It proposes several branching rules which are explained in [4]. Several tests have been done on the different instances but there is no rule that seems to be more effective than the others. So the used branching rule is the *reliability branching rule* defined in their paper.

For the node selector, the rule is the *best estimate search rule*, which is the default rule of SCIP.

2.8 Upper bound

The algorithm chosen for computing an upper bound of the optimal value is a tabu search. Several problems like job shop scheduling, graph coloring, travelling salesman problem and

other vehicle routing problems have been successfully tackled by the tabu search and for an introduction to this method see for instance [49]. While a greedy local search stops when no improving move is found in the neighborhood $N(s)$ of the current solution s, the basic principle of a tabu search is to continue the search by allowing non-improving moves. To avoid cycling on the same solution, the last ℓ moves are kept in the *tabu list* that is updated at each iteration. Its size ℓ is a parameter.

Various stopping criteria can be used, such for example: the predetermined number of iterations with no improvement, the maximum number of iterations, the maximum amount of time spent in the method. To define a tabu search properly we have to describe how to compute the cost of the current solution, the neighborhood, the way to encode the tabu list, a heuristic to build the first current solution and the stopping criterion.

2.8.1 Cost of the current solution

Proposition 2.3.1 allows to encode the solution as a sequence of vertices, starting and ending with the depot 0, without specifying bike displacements. The score of a sequence $i_1 \to i_2 \ldots \to i_k$ is the cost of the travelled distance $\sum_{j=1}^{k-1} c_{i_j i_{j+1}}$ to which we add a penalty when the sequence is infeasible, i.e. when there is no route inducing this sequence. The feasible sequence space may be disconnected, whence to explore efficiently the used neighborhood is enlarged allowing the tabu in visiting unfeasible solutions. Then, the infeasibility is penalized in the objective function. Thanks to Proposition 2.3.1 we are able to evaluate the infeasibility of any sequence indicated in the following with s. It is the "degree of infeasibility" we have defined in Section 2.3. The relaxed constraints are (2.11) and (2.12) and the resulting cost function $f(s)$ defined in (2.33) is inspired by the one proposed by [46] for the CVRP and is the following:

$$f(s) = \sum_{j=1}^{k-1} c_{(i_j, i_{j+1})} + \gamma \sum_{i \in V} (p_i - p'_i) \tag{2.33}$$

where γ is a positive constant and the p'_i are computed through the max-flow algorithm of Proposition 2.3.1, which imposes that $p'_i \leq p_i$.

In our experiment we have defined γ as 10 times the mean distance of a direct trip from the depot to a vertex. This choice is a way to convert at the right scale in term of cost the number of bikes that remain to be displaced (and was experimentally tuned).

2.8.2 Initial solution

We propose two distinct methods to compute the initial sequence needed for the tabu search.

Greedy heuristics

The method first tries to *close* vertices. Closing a vertex means bringing a vertex from its original state to its target state q_i in one move. The first criteria used, for deciding which vertex visit first, consists in ranking the unbalanced vertices with respect to their distance from the vehicle position and then in choosing the nearest node it can close. If it is impossible to close any vertex, then for each vertex the number of bikes the vehicle can load or unload is computed and the vehicle is driven toward the vertex with the highest number of bikes that can be exchanged: either loaded or unloaded. In case of equality between several vertices, it goes to the nearest one. With this method, we are sure to start from a feasible sequence.

The solution of (RP2)

A solution z of relaxation (RP2) of Section 2.5 is such that $G[z]$ is an Eulerian graph. A closed Eulerian path in the supported graph $G[z]$ provides a sequence of vertices that can be used as an initial solution of the tabu search. To compute such a closed Eulerian path various classical methods are available (see for instance page 31 of the book by [55]). Since the relaxation is quite good, this sequence is often not too far from an optimal solution. Note that if the sequence is feasible for the SVOCPDP, it is an optimal solution. Unfortunately, even if we have an instance for which the relaxation is tight, there is no polynomial algorithm for finding the optimal solution of the original problem (Proposition 2.6.3 above, Section 2.6).

2.8.3 Neighborhood description

At each iteration the whole neighborhood is explored. The definition of neighborhood is essential, since different neighborhoods offer different ways to explore the solution space but require different computational efforts. For solving the addressed problem, four different moves have been defined and used in the tabu search framework. Note that any time a vertex appears two consecutive times in a sequence one of the occurrences can be removed.

Let k denote the length of the sequence and note that $k >> n$, therefore is k instead of n that will be used for computing the complexity. To illustrate the different moves the solution displayed on Figure 2.2 will be used each time as initial solution.

$$0 \to 7 \to 6 \to 4 \to 5 \to 2 \to 8 \to 1 \to 0$$

Following this sequence, the vehicle leaves the depot to go to vertex 7. Then it goes on with vertex 6 and so on until vertex 1 from where it goes back to the depot.

2-OPT

This move is classical for routing problems. A pair of non-consecutive arcs are removed from the sequence and two new arcs are inserted. They link the two tails and the two heads of the removed arcs and therefore the travelling direction of the subset of nodes between the two removed arcs is reversed to obtain a new sequence. An example of 2-OPT is obtained by removing arcs (0,7) and (5,2), by inserting arcs (7,2) and (0,5) and reversing the sequence $5, 4, 6, 7$, therefore the following sequence is obtained:

$$0 \to 5 \to 4 \to 6 \to 7 \to 2 \to 8 \to 1 \to 0$$

This move is tested for each pair of edges in the sequence. It takes $O(k^2)$ to test all the moves of this type.

Suppression

This move tries to delete a vertex in the sequence. If in the example of Figure 2.2 we delete vertex 4, we obtain

$$0 \to 7 \to 6 \to 5 \to 2 \to 8 \to 1 \to 0$$

This move is tested for each vertex in the sequence. It takes $O(k)$ to test all moves of this type.

Add unbalanced vertex

This move is active when the current sequence is infeasible. All the vertices are checked and both the most unbalanced vertex i in excess and the most unbalanced vertex j in default are selected (vertices for which $|(q_i - q'_i) - (p_i - p'_i)|$ are maximal). One neighbor is obtained by adding, just after i in the sequence, the moves $\to j \to i \to$. The other neighbor is obtained by adding the moves $\to i \to j \to$ in the sequence just after j. If neither i nor j are present in the

sequence, then there is only neighbor obtained by adding $\rightarrow i \rightarrow j$ at the end of the sequence. Testing all moves of this type takes $O(k^2)$.

Once more with the example reported in Figure 2.2, if the capacity of the vehicle is lower than 8, the vertices are not balanced at the end of the sequence. Let $K = 6$, the vehicle cannot bring vertices 1, 6 and 8 to their target state: 2 bikes need to be loaded from 6 while 1 bike is requested by both node 1 and node 8. The moves $\rightarrow 8 \rightarrow 6$ are tried to be added and we obtain the following sequence

$$0 \rightarrow 7 \rightarrow 6 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 8 \rightarrow 1 \rightarrow 0$$

This sequence is a feasible solution (i.e. a route).

Add buffer vertex

This move tries to add a second time a vertex in the sequence. The second copy of a vertex can be used as buffer vertex, since in the buffer vertex bikes are either delivered or picked up. Thus, if a vertex is used as a buffer vertex it has to be visited at least twice. Every vertex can be used as buffer vertices. The sequence is traversed and at each position each vertex, that is already present, is tried to be inserted. If a vertex is not present in the sequence (for example the initially balanced vertices), then it is inserted twice.

This move is very time consuming and testing all moves of this type takes $O(k^3)$. Thus it is not called at each iteration but in 20% of the cases.

2.8.4 The tabu list

The fact that arcs could be used more than once leads to a management of the tabu list slightly different from the classical one used, for example, for solving the classical routing problem. When a move is made, some arcs leaves the solution and others enter the solution. Each exiting arc is kept in the tabu list along with the position of its occurrence in the last solution. This arc is forbidden for ℓ iterations, but it is allowed to be reinserted in a different position.

The tabu list is kept using the `multimap` structure in the STL of C++, which ensures a quick access.

2.8.5 The tabu search

The components previously described are integrated in the general algorithm summarized in Algorithm 1. The following notations are used in the pseudo-code: s is the current solution; s^* is the best feasible solution encountered from the beginning of the tabu search. $f(s)$ is the value of the objective function for the solution s; $NbIterMax$ is the maximum number of iterations done before the tabu search stops. i is the current iteration; $\bar{s}_X$ is the best solution encountered using the move X; $s^{\#}$ is the best solution in the whole neighborhood. a is a random variable uniformly distributed between 0 and 1.

2.9 Computational study

The algorithms have been coded in C++, embedded into SCIP (a Constraint Integer Programming framework, see [4]) and tested on a PC AMD Athlon 5600+ clocked at 2.8 GHz, with 16 GB RAM. The following Subsection 2.9.1 presents how the instances have been created, the results obtained are reported in Subsection 2.9.2 while Subsection 2.9.3 reports a discussion on the results.

2.9.1 Instances

The instances are taken from [67] which defines, for various values of n, 10 instances named from A to J, with an imbalance e_i between -10 and 10 for each vertex i. We work on these instances with $n \in \{20, 40, 60, 100\}$. In order to get an initial state p_i and a target state q_i for each vertex i, as required for the SVOCPDP, the demands have been modified. We do it twofold. A first set of instances has been obtained by setting $p_i = 10$, $q_i = 10 + e_i$ and $C_i = 20$ for each vertex i. We refer to these instances by the indication $\bar{p}_i = \bar{q}_i = 10$ (average state). A second set of instances is obtained by setting $p_i = 30$, $q_i = 30 + 3 \times e_i$ and $C_i = 60$ for each vertex i. We refer to these instances by the indication $\bar{p}_i = \bar{q}_i = 30$ (average state).

2.9.2 Computational results

The proposed algorithms have been tested on these instances for all $K \in \{10, 30, 45, 1000\}$. Note that even with huge capacity for the vehicle our problem is not a TSP with pickup and delivery. Initially balanced vertices can be skipped, nevertheless visiting a vertex several times can improve the cost of the solution (see for example the solution reported in Figures 2.4 and 2.5).

Algorithm 1 Tabu search algorithm

$s \leftarrow$ ComputeInitialSolution()
$s^* \leftarrow s$
$i \leftarrow 0$
while $i \leq NbIterMax$ **do**
  $\bar{s}_{2OPT} \leftarrow$ Explore2OPT(s)
  $\bar{s}_{Sup\leftarrow}$ ExploreSuppresion(s)
  $s^{\#} \leftarrow \text{argmin}(f(\bar{s}_{2OPT}), f(\bar{s}_{Sup}))$
  if s is not a route **then**
    $\bar{s}_{AddUnb} \leftarrow$ ExploreAddUnbalanced(s)
    if $f(s^{\#}) > f(\bar{s}_{AddUnb})$ **then**
      $s^{\#} \leftarrow \bar{s}_{AddUnb}$
    end if
  end if
  $a \leftarrow \text{Random}()$
  if $a \leq 0.2$ **then**
    $\bar{s}_{AddBuf} \leftarrow$ ExploreAddBuffer(s)
    if $f(s^{\#}) > f(\bar{s}_{AddBuf})$ **then**
      $s^{\#} \leftarrow \bar{s}_{AddBuf}$
    end if
  end if
  $s \leftarrow s^{\#}$ **and** Update the tabu list
  if s is a route and $f(s) < f(s^*)$ **then**
    $s^* \leftarrow s$
  end if
  $i \leftarrow i + 1$
end while

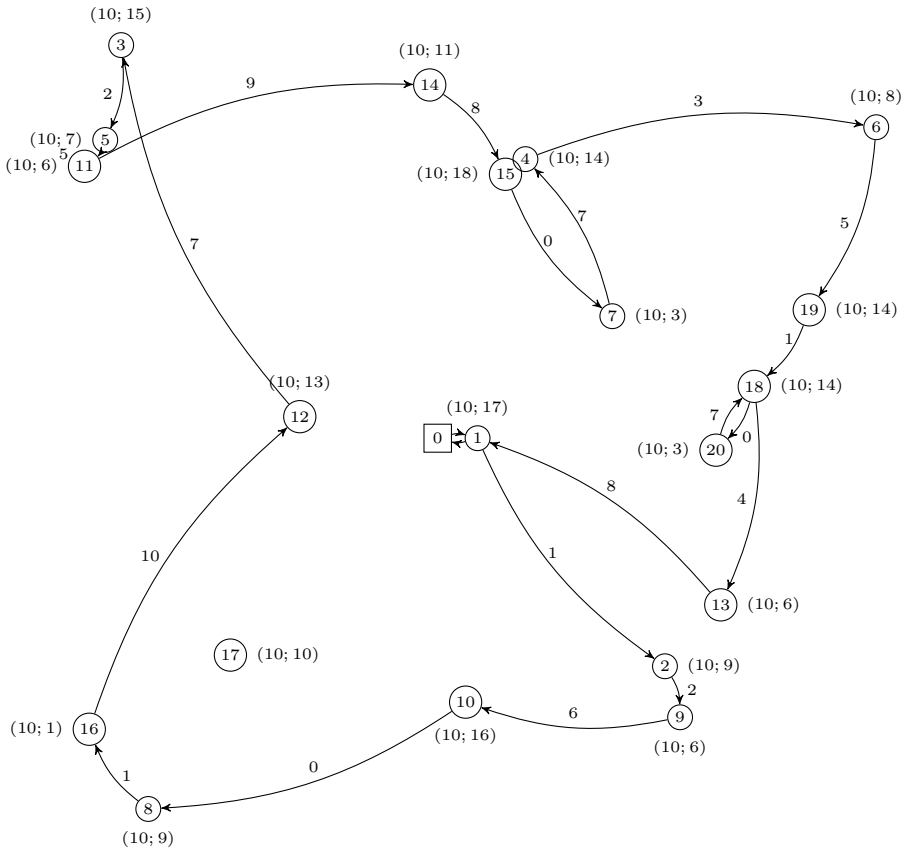

Figure 2.11: An optimal solution already found by the branch-and-cut for the instance n20G, with $K = 10$ and $\bar{p}_i = \bar{q}_i = 10$

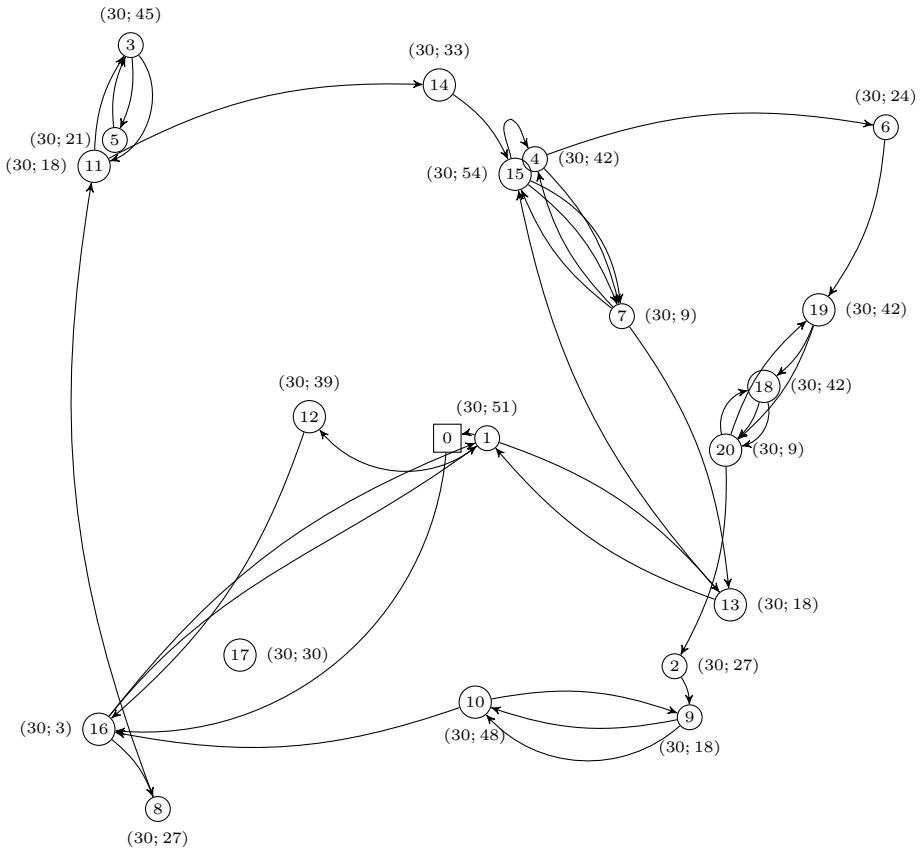

Figure 2.12: An optimal solution already found by the branch-and-cut for the instance n20G, with $K = 10$ and $\bar{p}_i = \bar{q}_i = 30$

The algorithms tested are the tabu search initialized with the greedy heuristics from Subsection 2.8.2, the branch-and-cut of Section 2.7 which provides a lower bound for the problem, and the tabu search initialized with the solution of the branch-and-cut as explained in Subsection 2.8.2.

The results are given in Tables 2.1, 2.2, 2.3 and 2.4. For each pair n, K, we have kept only the results for three instances: the best one, the worst one and the median one, with respect to the final gap. We have treated the results for $n = 100$ in distinct tables since the results obtained for such dimensions are less good than those obtained with $n < 100$.

The results achieved by all our algorithms when used for solving the generated instances are reported respectively in Tables 2.1, 2.3, 2.5 and 2.7. The tabu search initialized with the greedy heuristic (indicated in the following with TS1) has the following set of parameters: maximum number of iterations = $1'000$, the size of the tabu list (i.e. ℓ) = 30 and the maximum amount of time = $1'000$ seconds. The method also stops when no improvement has been done for 80 consecutive iterations. The branch-and-cut algorithm (indicated in the following with B&C) with the solution of TS1 for upper bound has for time limit $10'000$ seconds. The tabu search initialized with the solution obtained by the branch-and-cut B&C (indicated in the following with TS2) has the following parameters: the tabu list size is set to 3, the maximal number of iteration to 50 and the maximal amount of time to 300 seconds. The research is stopped whenever 15 consecutive iterations did not improve the best solution or if a solution is found with a cost equal to the lower bound obtained by the branch-and-cut. The abbreviations used in these tables are the following:

n is the number of vertices.

K is the vehicle capacity.

UB1 is the value obtained by TS1.

Time is the cpu time (in seconds) used by the Tabu Search TS1.

Iter. is the number of iterations.

UB2 is the best solution obtained by the tabu searches TS1 and TS2.

T.t. is the cpu time (in seconds) elapsed for executing TS1+B&C+TS2.

LB is the best lower bound found by the branch-and-cut.

Gap % is the percentage gap between the previous LB and UB2.

The results achieved by the branch-and-cut alone are reported respectively in Tables 2.2, 2.4, 2.6 and 2.8. The abbreviations used in these tables are the following:

n is the number of vertices.

K is the vehicle capacity.

UB-BC is the best solution found by the branch-and-cut.

LB is a lower bound on the optimal solution of the relaxation.

LBr is the lower bound that has been found at the root node.

Gap % is the percentage gap between UB-BC and LB.

Time is the computational time.

 # Nodes is the number of nodes of the branch-and-cut tree.

Figure 2.11 gives the solution for the instance n20G with $K = 10$, and an average number of bikes per station equaling 10. The depot appears as a square with a 0 and next to each vertex (p_i, q_i) is given in parenthesis. The solution displayed is the one obtained at the end of the following sequence of algorithms. The tabu search first, then the branch-and-cut with the result of the tabu search result as upper bound. Finally, the tabu search which starts from the solution given by the branch-and-cut. The cost of this route is displayed in the upper box. In this example the route displayed is exactly the solution of the branch-and-cut. It is the optimal solution even if a vertex (vertex 18) is visited twice. Compare with Figure 2.12. It is the same instance, but this time with an average number of bikes per station equaling 30. Again, the branch-and-cut was able to find the optimal solution. Interestingly, we can see that, with an increasing average number of bikes but with the same vehicle capacity, optimal solutions may visit vertices several times.

2.9.3 Conclusion

As expected the computational time and the number of nodes of the branch-and-cut algorithm increase with the number of vertices and decrease with the capacity of the vehicle. In Tables 2.2, 2.4, 2.6 and 2.8, the number of nodes used in the branch-and-cut could seem really huge. However, since variables are integers (not binary), several branchings must be done on the same variable before obtaining the optimal solution. Except for the large instances with 100 vertices, the results reported in Tables 2.1 and 2.5 show that the distance between the best upper bound found and the lower bound is really small. The gap is on average less that 5%. The local search is very efficient on small and medium instances (up to 60 vertices) and becomes

Table 2.1: Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$, less than 60 vertices, and a vertex capacity equal to 20

Instance name	**n**	**K**	**UB1**	**Time**	**Iter.**	**UB2**	**T.t.**	**LB**	**Gap %**
n20A	20	10	4897	6	104	4702	7	4702.00	0.00
n20C	20	10	6188	12	214	6013	14	6012.00	0.02
n20B	20	10	5009	8	149	4769	8	4769.00	0.00
n20A	20	30	3583	3	89	3583	4	3583.00	0.00
n20E	20	30	4556	4	88	4556	5	4299.00	5.98
n20F	20	30	4422	5	97	4108	5	4108.00	0.00
n20A	20	45	3583	4	89	3583	4	3583.00	0.00
n20C	20	45	4045	6	87	4045	7	3891.00	3.96
n20B	20	45	3792	6	108	3792	6	3792.00	0.00
n20A	20	1000	3583	4	89	3583	4	3583.00	0.00
n20C	20	1000	4045	4	87	4045	5	3891.00	3.96
n20J	20	1000	3801	7	138	3650	7	3650.00	0.00
n40E	40	10	6870	208	112	6424	2253	6424.00	0.00
n40F	40	10	7234	460	243	7095	10509	6760.00	4.96
n40J	40	10	7228	144	95	6268	10067	6267.00	0.02
n40A	40	30	5419	176	104	4949	178	4949.00	0.00
n40C	40	30	4806	409	217	4692	450	4644.00	1.03
n40B	40	30	5272	296	167	5110	301	5110.00	0.00
n40E	40	45	5378	184	123	5069	188	5069.00	0.00
n40H	40	45	5008	174	120	5006	207	4772.00	4.90
n40D	40	45	5604	297	166	5202	326	5195.00	0.13
n40A	40	1000	5042	144	90	4949	153	4949.00	0.00
n40C	40	1000	4763	246	142	4656	284	4522.00	2.96
n40B	40	1000	5297	153	90	5110	159	5110.00	0.00
n60H	60	10	10126	1039	60	8208	11328	7707.44	6.49
n60B	60	10	9382	975	33	8723	11312	7508.53	16.17
n60A	60	10	9500	1006	42	8010	11349	7276.80	10.08
n60G	60	30	6626	988	47	6360	1264	6360.00	0.00
n60I	60	30	7242	984	55	6766	8234	6390.00	5.88
n60H	60	30	6578	1037	41	6081	1835	5992.00	1.49
n60J	60	45	6601	1002	44	6374	1112	6374.00	0.00
n60D	60	45	6866	988	56	6462	1335	6252.00	3.36
n60B	60	45	7098	1015	32	6132	1321	6054.00	1.29
n60D	60	1000	6903	1049	41	6223	1395	6223.00	0.00
n60F	60	1000	6524	975	37	6136	1540	5778.00	6.20
n60C	60	1000	6258	988	48	6218	1285	6114.00	1.70

Table 2.2: Performance of the branch-and-cut cited in Table 2.1

Instance name	n	K	UB-BC	LB	LBr	Gap %	Time	# Nodes
n20A	20	10	4702	4702.00	4618.10	0.00	1.43	165
n20C	20	10	6012	6012.00	5833.33	0.00	1.54	199
n20B	20	10	4769	4769.00	4735.00	0.00	0.58	29
n20A	20	30	3583	3583.00	3583.00	0.00	0.10	1
n20E	20	30	4299	4299.00	4299.00	0.00	0.17	1
n20F	20	30	4108	4108.00	4108.00	0.00	0.13	1
n20A	20	45	3583	3583.00	3583.00	0.00	0.10	1
n20C	20	45	3891	3891.00	3891.00	0.00	0.04	1
n20B	20	45	3792	3792.00	3792.00	0.00	0.13	1
n20A	20	1000	3583	3583.00	3583.00	0.00	0.13	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.12	1
n200J	20	1000	3650	3650.00	3650.00	0.00	0.28	1
n40E	40	10	6424	6424.00	5968.25	0.00	2044.07	124799
n40F	40	10	7102	6760.00	6558.40	5.06	10000.00	688473
n40J	40	10	6267	6267.00	5655.58	0.00	9894.40	704853
n40A	40	30	4949	4949.00	4934.50	0.00	1.91	13
n40C	40	30	4644	4644.00	4644.00	0.00	1.17	1
n40B	40	30	5110	5110.00	4941.63	0.00	4.47	84
n40E	40	45	5069	5069.00	5032.50	0.00	4.12	32
n40H	40	45	4772	4772.00	4754.00	0.00	1.47	3
n40D	40	45	5195	5195.00	5195.00	0.00	1.90	1
n40A	40	1000	4949	4949.00	4922.00	0.00	3.47	7
n40C	40	1000	4522	4522.00	4522.00	0.00	2.42	1
n400B	40	1000	5110	5110.00	5040.50	0.00	5.56	47
n60H	60	10	8208	7707.44	7606.03	6.49	10000.02	133440
n60B	60	10	9367	7508.53	7370.35	24.75	10000.00	128883
n60A	60	10	8397	7276.80	7195.56	15.39	10000.01	119974
n60G	60	30	6360	6360.00	6212.40	0.00	266.02	4727
n60I	60	30	6390	6390.00	6042.50	0.00	6954.05	138870
n60H	60	30	5992	5992.00	5665.75	0.00	514.69	8615
n60J	60	45	6374	6374.00	6246.81	0.00	103.94	1139
n60D	60	45	6252	6252.00	6213.40	0.00	166.21	1574
n60B	60	45	6054	6054.00	6018.50	0.00	17.28	76
n60D	60	1000	6223	6223.00	6177.33	0.00	339.27	320
n60F	60	1000	5778	5778.00	5726.00	0.00	258.91	534
n60C	60	1000	6114	6114.00	6068.50	0.00	25.02	27

Table 2.3: Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$ and 100 vertices of capacity 20

Instance name	n	K	UB1	Time	Iter.	UB2	T.t.	LB	Gap %
n100E	100	10	16086	1125	5	12895	11333	10175.27	26.73
n100C	100	10	20152	971	2	16239	11234	11707.18	38.71
n100H	100	10	15951	949	3	14507	11367	10985.15	32.06
n100A	100	30	11962	1331	5	8245	11596	7575.25	8.84
n100D	100	30	13013	1075	1	12284	11290	7913.46	55.23
n100J	100	30	11113	1056	1	8863	12511	7447.41	19.01
n100G	100	45	10956	1052	1	7909	8149	7860.00	0.62
n100J	100	45	9644	1071	1	8539	11334	7159.86	19.26
n100C	100	45	9688	1215	4	8630	11465	7779.75	10.93
n100E	100	1000	8636	1681	10	8065	12488	7680.50	5.01
n100B	100	1000	8940	1153	3	8940	11368	7159.50	24.87
n100D	100	1000	9067	1603	7	7958	11900	7294.00	9.10

Table 2.4: Performance of the branch-and-cut cited in Table 2.3

Instance name	n	K	UB-BC	LB	LBr	Gap %	Time	# Nodes
n100E	100	10	13449	10175.27	10168.96	32.17	10000.01	28483
n100C	100	10	18107	11707.18	11691.72	54.67	10000.03	21190
n100H	100	10	15752	10985.15	10978.65	43.39	10000.01	29422
n100A	100	30	8133	7575.25	7504.21	7.36	10000.00	38885
n100D	100	30	9420	7913.46	7887.63	19.04	10000.04	34007
n100J	100	30	9136	7447.41	7409.02	22.67	10000.07	39026
n100G	100	45	7860	7860.00	7696.40	0.00	6855.28	26349
n100J	100	45	8026	7159.86	7063.07	12.10	10000.01	35323
n100C	100	45	8458	7779.75	7751.26	8.72	10000.05	31024
n100E	100	1000	8006	7680.50	7672.21	4.24	10000.35	1527
n100B	100	1000	8619	7159.50	7153.19	20.39	10002.70	864
n100D	100	1000	8054	7294.00	7280.96	10.42	10000.90	863

Table 2.5: Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$, less than 60 vertices and a vertex capacity equal to 60

Instance name	**n**	**K**	**UB1**	**Time**	**Iter.**	**UB2**	**T.t.**	**LB**	**Gap %**
n20B	20	10	9888	70	182	9883	71	9883.00	0.00
n20C	20	10	14127	52	101	14040	137	14039.00	0.01
n20D	20	10	15365	160	136	14925	247	14925.00	0.00
n20B	20	30	4808	15	222	4769	16	4769.00	0.00
n20C	20	30	6362	19	233	6013	23	6012.00	0.02
n20D	20	30	6252	15	216	5989	16	5989.00	0.00
n20B	20	45	4175	6	98	4174	6	4174.00	0.00
n20C	20	45	5295	6	89	5295	9	5113.00	3.56
n20D	20	45	5533	8	99	5446	12	5446.00	0.00
n20B	20	1000	3792	6	102	3792	6	3792.00	0.00
n20C	20	1000	4098	8	133	4045	9	3891.00	3.96
n20J	20	1000	3763	8	174	3650	8	3650.00	0.00
n40E	40	10	13708	1013	46	13159	1786	13159.00	0.00
n40F	40	10	15448	1011	53	15410	11309	14456.90	6.59
n40I	40	10	15722	989	66	14849	2531	14849.00	0.00
n40E	40	30	6670	271	152	6424	1024	6424.00	0.00
n40F	40	30	7653	209	114	7240	10239	6571.83	10.17
n40I	40	30	7045	342	175	6901	2144	6901.00	0.00
n40A	40	45	6484	182	95	6059	492	6059.00	0.00
n40B	40	45	5665	345	191	5319	357	5319.00	0.00
n40C	40	45	6095	245	142	5912	477	5912.00	0.00
n40B	40	1000	5253	215	148	5110	221	5110.00	0.00
n40C	40	1000	4684	418	269	4656	439	4522.00	2.96
n40E	40	1000	5441	217	136	5069	228	5069.00	0.00
n60F	60	10	23791	1133	4	17696	11414	16925.71	4.55
n60A	60	10	24684	904	1	18755	11075	15789.56	18.78
n60J	60	10	28130	915	1	17462	11136	15774.62	10.70
n60H	60	30	8980	1036	47	8120	11334	7608.96	6.72
n60C	60	30	10367	992	61	9818	11227	8313.06	18.10
n60J	60	30	9049	1072	54	8407	11357	7642.33	10.01
n60H	60	45	7333	998	55	6743	2986	6743.00	0.00
n60D	60	45	9084	1009	58	8778	11298	7631.48	15.02
n60F	60	45	8804	986	58	7134	11310	6731.50	5.98
n60D	60	1000	6685	993	90	6223	2390	6223.00	0.00
n60F	60	1000	6539	989	63	6136	1266	5778.00	6.20
n60A	60	1000	5982	1028	49	5889	1191	5740.00	2.60

Table 2.6: Performance of the branch-and-cut cited in Table 2.5

Instance name	**n**	**K**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n20B	20	10	9883	9883.00	9872.33	0.00	0.32	9
n20C	20	10	14039	14039.00	13783.63	0.00	73.23	33867
n20D	20	10	14925	14925.00	14594.67	0.00	86.23	58991
n20B	20	30	4769	4769.00	4698.00	0.00	1.36	37
n20C	20	30	6012	6012.00	5750.00	0.00	3.50	375
n20D	20	30	5989	5989.00	5968.25	0.00	1.05	25
n20B	20	45	4174	4174.00	4174.00	0.00	0.22	1
n20C	20	45	5113	5113.00	5035.00	0.00	2.92	423
n20D	20	45	5446	5446.00	5174.00	0.00	3.01	381
n20B	20	1000	3792	3792.00	3792.00	0.00	0.15	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.09	1
n20J	20	1000	3650	3650.00	3650.00	0.00	0.44	1
n40E	40	10	13159	13159.00	12575.48	0.00	761.07	59441
n40F	40	10	15448	14456.90	14256.09	6.86	10000.00	723945
n40I	40	10	14849	14849.00	14386.27	0.00	1531.88	115415
n40E	40	30	6424	6424.00	6075.50	0.00	751.60	40129
n40F	40	30	7239	6571.83	6348.80	10.15	10000.00	702288
n40I	40	30	6901	6901.00	6447.67	0.00	1801.81	119722
n40A	40	45	6059	6059.00	5807.12	0.00	305.81	23153
n40B	40	45	5319	5319.00	5211.50	0.00	11.00	462
n40C	40	45	5912	5912.00	5530.60	0.00	231.63	14986
n40B	40	1000	5110	5110.00	5040.50	0.00	5.28	33
n40C	40	1000	4522	4522.00	4522.00	0.00	2.27	1
n40E	40	1000	5069	5069.00	5032.50	0.00	11.44	55
n60F	60	10	18098	16925.71	16889.78	6.93	10000.00	176987
n60A	60	10	19403	15789.56	15691.49	22.88	10000.00	134007
n60J	60	10	17929	15774.62	15674.85	13.66	10000.00	175201
n60H	60	30	8378	7608.96	7411.93	10.11	10000.02	134581
n60C	60	30	10185	8313.06	8211.90	22.52	10000.02	156541
n60J	60	30	8408	7642.33	7518.03	10.02	10000.00	132487
n60H	60	45	6743	6743.00	6544.58	0.00	1979.12	29674
n60D	60	45	9061	7631.48	7534.96	18.73	10000.00	128581
n60F	60	45	7202	6731.50	6688.64	6.99	10000.00	166326
n60D	60	1000	6223	6223.00	5868.42	0.00	1394.29	3364
n60F	60	1000	5778	5778.00	5726.00	0.00	112.45	343
n60A	60	1000	5740	5740.00	5607.25	0.00	58.04	99

Table 2.7: Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$ and 100 vertices of capacity 60

Instance name	n	K	UB1	Time	Iter.	UB2	T.t.	LB	Gap %
n100B	100	10	47117	2937	1	32891	62810	27359.37	20.22
n100H	100	10	44434	1427	1	34989	98532	26254.29	33.27
n100F	100	10	43570	1112	1	28673	55675	23402.08	22.52
n100A	100	30	15403	1867	6	12996	25638	10177.64	27.69
n100I	100	30	19164	651	1	16190	29729	11567.98	39.96
n100G	100	30	16233	1069	3	13472	26420	10251.04	31.42
n100A	100	45	12615	1617	4	10374	22850	8321.17	24.67
n100B	100	45	16118	1398	2	12726	26505	9185.89	38.54
n100H	100	45	13349	1419	8	12037	23518	9221.84	30.53
n100E	100	1000	9067	969	2	8071	19351	7700.00	4.82
n100A	100	1000	9550	1441	2	8822	11595	6969.00	26.56
n100J	100	1000	8178	1532	7	8138	19357	6918.68	17.62

Table 2.8: Performance of the branch-and-cut cited in Table 2.7

Instance name	n	K	UB-BC	LB	LBr	Gap %	Time	# Nodes
n100B	100	10	34964	27359.37	27295.96	27.80	10000.02	15032
n100H	100	10	37156	26254.29	26213.62	41.52	10000.04	14142
n100F	100	10	30489	23402.08	23365.92	30.28	10000.10	22153
n100A	100	30	13686	10177.64	10171.43	34.47	10000.01	24026
n100I	100	30	17210	11567.98	11522.04	48.77	10000.15	26068
n100G	100	30	14087	10251.04	10234.99	37.42	10000.01	30956
n100A	100	45	10744	8321.17	8260.51	29.12	10000.06	29399
n100B	100	45	12624	9185.89	9141.17	37.43	10000.04	31747
n100H	100	45	12645	9221.84	9212.68	37.12	10000.02	42640
n100E	100	1000	7700	7700.00	7498.67	0.00	1256.14	4477
n100A	100	1000	8747	6969.00	6957.76	25.51	10000.00	1065
n100J	100	1000	8178	6918.68	6911.71	18.20	10000.90	1643

less effective for larger instances. This is probably a consequence of the size of the neighborhood, which can be quite huge when the vehicle has to make several visits at some vertices. For $n = 100$, the second tabu search makes sometimes only one or two iterations.

Note also that the smaller is the capacity, the harder is the problem. It is in accordance with the intuition, since for instances with small vehicle capacity, the mean number of visits by vertex increases. Instances n20A and n40E in Table 2.1 are illustrations for such a phenomenon: compare the optimal results for $K = 10$ and $K = 45$. We have also such a phenomenon for n20B and n40E in Table 2.5 for $K = 10$ and $K = 30$.

If we go back to the size of the Vélib' system in Paris (one truck of capacity 20 for 60 vertices of capacity 30), the instances that are close to these numerical features are the instances n60G, n60H, n60I of Table 2.1 with $n = 60$ and $K = 30$ and the instances n40E, n40F, n40I of Table 2.5 with $n = 40$ and $K = 30$. We get solutions that have most of time a gap smaller than 2%. It would certainly be possible to improve the time consumption of the flow algorithm in the tabu search. Stopping the branch-and-cut after a fixed time would also be a reasonable solution to reduce the total time to get a good solution. However, we see that 50% of the instances of this size can already be solved within optimality gap of 2% in less than 40 minutes.

Chapter 3

The Multiple-Vehicle Balancing Problem

3.1 Introduction

The Multiple-Vehicle Balancing Problem (MVBP) is the multiple-vehicle variant of the SVOCPDP. Here also users' impact on the system is neglected and the objective is to move bikes from stations to other in order to fit a given repartition which is known for being suitable to answer the morning peak. To that purpose, several vehicles are available and the operator has to provide instructions to the drivers in order to bring the system to the target repartition while minimizing the total distance travelled by the whole fleet of vehicles. This problem has similarities with pickup-and-delivery problems and with the Split Delivery Vehicle Routing Problem (SDVRP) as explained in detail in Section 3.3.

We model this problem as an integer program (IP) whose variables encode the selection of possible routes. The relaxation of the program – called the *master problem* – is solved with a column generation method. Such a method is often used to solve problems with a huge number of variables. Instead of considering all of them, the original problem is only solved on a subset of variables - or columns - forming the *restricted master problem*. A *pricing subproblem* uses the dual variables of the restricted master problem and is solved to obtain the columns that improve the value of the objective function of the restricted master problem. Once we know how to solve the restricted master problem, there are two ways for solving the original IP. The first one is to perform a branch-and-price algorithm (see for example [13] for a survey on this topic). The second one is to add all columns that may play a role in the optimal of the IP. These "columns in the gap" can be identified thanks to their reduce costs. They are then added and the resulting integer program is solved with the help of CPLEX (see for example [11] for a

reference on this approach). This method requires to have a good feasible solution. Since we have chosen to follow such an approach, we have designed an efficient memetic algorithm to perform this task. This algorithm is based on a combinatorial encoding of the solutions which does not need to specify the loading and unloading operations, and uses the "giant tour" idea proposed by Prins [73].

The problem is described in Section 3.2. A short overview of the literature devoted to this topic is presented in Section 3.3. The next section – Section 3.4 – is devoted to the modelling of the problem. Some dominances are proven allowing to derive a set partitioning-like model. Section 3.5 focuses on the relaxation and formalizes the master problem as well as the pricing subproblem. Section 3.6 explains how to solve the pricing subproblem. In Section 3.7, with the help of dual-feasible functions, additional cuts are defined to enhance the linear relaxation of the model. The memetic algorithm finding a good feasible solution is described in Section 3.8. Computational results are given in Section 3.9.

3.2 Problem and notations

3.2.1 Problem definition

The problem, which we call the *Multiple-Vehicle Balancing Problem* can be formalized as follows: let $G = (V, A)$ be a complete directed graph in which $V = \{0, \ldots, n\}$ is the vertex set composed by $n+1$ vertices, the vertices in $\{1, \ldots, n\}$ representing the stations and the vertex 0 representing the depot and A is the set of arcs. For each arc $(i, j) \in A$, we denote by c_{ij} the cost of the arc (i, j). Each arc corresponds to the shortest path between its tail vertex and its head vertex. The cost is assumed to satisfy the triangular inequality (i.e. $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$). Each vertex i has a capacity $C_i \in \mathbb{Z}_+$. For each $i \in V$, we define its initial state by $p_i \in \mathbb{Z}_+$ and its target state $q_i \in \mathbb{Z}_+$, with $p_i \leq C_i$, $q_i \leq C_i$ and $\sum_{i \in V} p_i = \sum_{i \in V} q_i$. Throughout the chapter we use the *demand* $d_i = |p_i - q_i|$, number of bikes to be either bring or taken from a vertex. The depot is always assumed to have no bike: $p_0 = q_0 = 0$ and $d_0 = 0$. When $p_i > q_i$ (resp. $p_i < q_i$) the vertex is a *pickup* (resp. *delivery*) vertex. Vertices such that $p_i = q_i$ are called *initially balanced* vertices. $\mathcal{P}i$ denotes the set of pickup vertices and $\mathcal{D}e$ the set of delivery vertices. For all vertices $i, j \in V$ such that neither i nor j are initially balanced vertices, (i, j) is either a *pickup-to-pickup, delivery-to-delivery, pickup-to-delivery* or *delivery-to-pickup* arc with respect to the type of vertices i and j are. A homogeneous fleet of

M vehicles with a capacity $K \in \mathbb{Z}_+$ is given. Each vehicle leaves the depot, visits a sequence of vertices where it performs pickup or delivery services and returns to the depot. Each vehicle leaves at most once the depot. The objective is to redistribute the bikes in order to reach the target state priorly defined while minimizing the total distance travelled by the whole fleet of vehicles.

We assume moreover that there is a bound on the number of arcs each vehicle traverses. This bound is denoted $Rmax \in \mathbb{Z}_+$. This bound enables to limit the duration of a tour. This is all the more true if the loading and unloading time is important with respect to the travel time, which occurs often in practice. Tasks have to be distributed among vehicles. Without such a bound, it would not be rare to face optimal solutions with only one vehicle operating: for instance, if the depot is far away from the other vertices. Note that if $Rmax < \frac{\sum_{i \in V} d_i}{MK}$, there is no feasible solution to the problem.

To avoid logistic problem, the convergence to the target state is required to be monotonous: bikes can only be picked-up (resp. delivered) at pickup (resp. delivery) vertices. In particular, drops are not allowed. Without monotonous convergence, situations where a vehicle arrive at an empty vertex where it is supposed to load bikes could occur. This requirement enables to deal with each tour independently. One vertex can still be visited several times by the same vehicle or by different vehicles. Note that another consequence of the convergence requirement is that initially balanced vertices are not visited by any vehicle and so from now on they are not considered.

The MVBP contains the Travelling Salesman Problem (TSP), and is therefore $\mathcal{NP}$-hard. Indeed, if $Rmax = n + 1$, $K = n$ and there is a unique pickup vertex at a distance 0 from the depot with an excess of $n - 1$ bikes and all the $n - 1$ other vertices are delivery ones with a demand of 1 bike, then the solution of the TSP on the $n - 1$ delivery vertices to which a first stop at the pickup vertex is added to load all the bikes is the optimal solution of the MVBP.

Moreover it contains the SDVRP which is known to be harder than the TSP. In the SDVRP, goods have to be distributed to several customers. A fleet of vehicles is available at a depot with a given capacity. The objective is to find a collection of routes to fulfill each customer's demand while minimizing the total distance travelled by all vehicles. Customers can be visited by several vehicles as their demand can be satisfied in one or more visits. If there is an instance with only one pickup vertex at zero-distance from the depot with all the bikes and such that all the other vertices are delivery vertices and if $Rmax \geq K + 1$, then the SDVRP on the delivery vertices to which is added a stop at the pickup vertex for each route is equivalent to the MVBP.

3.2.2 Notations

Let $\mathcal{R}$ be the set of all routes. A route $r \in \mathcal{R}$ is a sequence of visited vertices associated with the number of picked up or delivered bikes at each stop. It is referred to as a couple (s_r, l_r), where $s_r = (s_{1r}, \ldots, s_{tr})$ is the sequence of visited vertices and $l_r = (l_{1r}, \ldots, l_{tr})$ the number of picked up or delivered bikes at each stop, with $t < Rmax$. The cost of the route c_r is defined as the total travelled distance while following its sequence. To each route $r \in \mathcal{R}$ and each vertex $i \in V$ is associated $b_{ir} \in \mathbb{Z}_+$, the total number of bikes picked up or delivered at vertex i in route r. Note that according to the convergence requirement, we necessarily have $b_{ir} \leq d_i$ for all $i \in V$, $r \in R$. Figure 3.1 shows an example whith 4 vertices and a depot. Vertices 1 and 3 (respectively 2 and 4) are pickup vertices (resp. delivery vertices). A truck following the route r displayed starts from the depot and first visits vertex 1 where 3 bikes are loaded, visits then vertex 4 where 2 bikes are unloaded, goes back to vertex 1 to load 2 other bikes before getting to vertex 2 to unload 3 bikes and driving back to the depot. The number of bikes loaded at each vertex appears next to the arrow (if this number is negative, it means that bikes are unloaded). In that case, we have for this route $s_r = (1, 4, 1, 2)$ and $l_r = (3, 2, 2, 3)$, $b_{1r} = 5$, $b_{2r} = 3$, $b_{3r} = 0$ and $b_{4r} = 2$. Note $l_{ri} \geq 0$ for all $i < Rmax$: bikes are loaded or unloaded with respect to the type of the vertex because of the monotonous convergence.

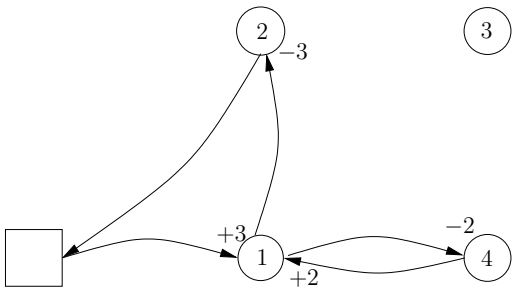

Figure 3.1: Example of the route

3.3 Literature

The MVBP has some similarities with the SDVRP – as it has already been noted at the end of the Section 3.2.1. Dror and Trudeau [35] are the first to introduce this problem, They prove that the split dimension of the problem could lead to huge savings in term of the objective function. They show that some properties hold for the optimal solution of the SDVRP

(dominances). However, the split component of the problem increases the complexity of the algorithm as explained by Archetti and Speranza [8]. Archetti *et al.* [7] present a tabu search algorithm to solve this problem. A few years after, Archetti *et al.* [9] propose an exact column generation method. Their branch-and-price-and-cut algorithm manage to find the optimal solution for medium size instances. Beside one with 144 vertices, they manage to find the optimal solution for instances with up to 48 vertices in 6 hours. These instances can be solved with the method we propose for the MVBP, as explained at the end of the former section. In Section 3.9, the method is tested on some of the instances solved by Archetti *et al.* [9].

Problems presenting similar features with ours are pickup-and-delivery problems, for which an extensive literature exist. One of the most powerful exact approach for a version with time windows (PDPTW) is the one proposed by Baldacci *et al.* [10]. The bound $Rmax$ is reminiscent of these time windows as it prevents the assignment of all tasks to a unique vehicle, and their approach will be used as a backbone for our method.

3.4 Dominance properties, model and method

3.4.1 Dominance properties

"Dominance properties" are properties satisfied by some optimal solutions. Translating them as new constraints for the problem does not modify the optimal value but reduces the number of feasible solutions.

Proposition 3.4.1. *Any optimal solution with a minimal number of vertex visits is such that, for any pair of vertices $\{i, j\}$ with i and j both pickup vertices or both delivery vertices, the number of times arc (i, j) is used plus the number of times arc (j, i) is used is at most one.*

Say that there is a *traversal of* $\{i, j\}$ each time there is a traversal of (i, j) or of (j, i) by a route. Proposition 3.4.1 states that the total number of traversals by all routes for such solutions on any $\{i, j\}$ with i and j of same type is at most 1.

Proof of Proposition 3.4.1. Let i and j be two pickup vertices.

Suppose that we have an optimal solution a minimal number of vertex visits and at least two traversals of $\{i, j\}$. Number these two traversals 1 and 2. Denote a_1 (resp. a_2) the number of bikes loaded at i during traversal 1 (resp. 2). Denote b_1 (resp. b_2) the number of bikes loaded at j during traversal 1 (resp. 2). Two cases have to be considered.

- **Case** $a_2 \geq b_1$**.** We can take $a_1 + b_1$ bikes at i instead of taking only a_1 bikes and avoid j for traversal 1. The remaining of the solution does not change (vertices visited and actions remain the same) except for traversal 2: we take $a_2 - b_1$ bikes on i and $b_1 + b_2$ bikes on j.
- **Case** $a_2 < b_1$**.** We can take $a_1 + a_2$ bikes at i instead of taking only a_1 bikes and take $b_1 - a_2$ bikes at j for traversal 1. The remaining of the solution does not change (vertices visited and actions remain the same) except for traversal 2. We avoid then i, goes directly to j where we load $a_2 + b_2$ bikes.

In both cases, we are able to reduce by one the number of traversals on $\{i, j\}$ without changing the remaining of the solution. This is in contradiction with its minimality.

The conclusion is similar if both i and j are delivery vertices. □

Let us consider the following two properties a solution may satisfy.

Property 1. *For any pickup-to-delivery arc (i, j), there is at most one route and at most one traversal of this arc by the route with a load strictly less than K.*

For any delivery-to-pickup arc (i, j), there is at most one route and at most one traversal of this arc by the route with a load strictly greater than 0.

Property 2. *For any pickup vertex i and any delivery vertex j, if there is at least one traversal of (i, j) and one traversal of (j, i), then all traversals of (i, j) are done with K bikes.*

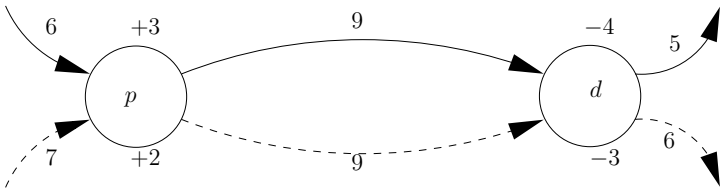

Figure 3.2: Assume $K = 10$. The solid line route and the dotted line route are both having a vehicle travelling the arc (p, d) without being full. Transferring one picked-up bike from one of the route to the other, we see that a move between p and d can be done with 10 bikes – which is the vehicle capacity – and the other with 8 bikes without changing the impact on the vertices

Proposition 3.4.2. *Given an optimal solution with a minimal number of vertex visits, either it satisfies simultaneously Property 1 and Property 2, or there is another one with the same sequence of vertices satisfying simultaneously Property 1 and Property 2*

Proof. The proof consists in showing that one can modify the load of the trucks in any route of an optimal solution locally, without changing the remaining, in order to satisfy each of the two properties. Let us consider an optimal solution with a minimal number of vertex visits.

Property 1

Let i be a pickup vertex and j be a delivery one. Assume that the solution visits at least twice the arc (i, j), each time with strictly less than K bikes. Transferring bikes between these two traversals provides a traversal with K bikes (we can not get a feasible solution with 0 bike on one of these traversals since the solution is assumed to be optimal). See Figure 3.2 for an illustration.

We get a similar conclusion if i is a delivery vertex and j a pickup one.

Property 2

We assume that the optimal solution satisfies Property 1. Suppose that there is a traversal of (i, j) with strictly less than K bikes and a traversal of (j, i), with i a pickup vertex and j a delivery vertex. The solution being optimal, during the traversal (j, i), there is at least one bike unloaded on j and one bike loaded on i. This bike could be taken from i and unloaded on j during the unsaturated traversal of (i, j). We can repeat this remark until there are K bikes carried during the traversal of (i, j). □

3.4.2 The model

We recall that a route is the sequence of visited vertices with the loading and unloading operations at each vertex. Given the set of all routes $\mathcal{R}$, we can write the following set partitioning-like formulation of the problem (SPF):

$$z(SPF) = \min \sum_{r \in \mathcal{R}} c_r z_r \tag{3.1}$$

$$\text{s.t.} \sum_{r \in R} b_{ir} z_r = d_i \quad \forall i \in V \tag{3.2}$$

$$\sum_{r \in R} z_r \leq M \tag{3.3}$$

$$z_r \in \{0, 1\} \quad \forall r \in \mathcal{R} \tag{3.4}$$

where z_r is a binary variables equal to 1 if route $r \in \mathcal{R}$ is in the solution, 0 otherwise.

At first glance, z_r might take values equal to or larger than 2 in an optimal solution: a route may be selected several times. However, according to Section 3.4.1, we see that if $d_i \leq K$ for all i, we may assume that each route is taken at most once. Indeed, Proposition 3.4.1 shows that two distinct routes present in an optimal solution can not use both the same pickup-to-pickup or delivery-to-delivery arc. Thus, if a route is used several times in an optimal solution, then it consists in a alternate sequence of pickup and delivery vertices. Proposition 3.4.2 shows moreover that such a route has a load of the vehicle equal to K for a pickup-to-delivery arc and equal to 0 for a delivery-to-pickup arc. Therefore, if $d_i \leq K$ for all i, such a route can be used at most once.

If there are vertices i with $d_i > K$, we could split them into as many vertices as necessary to get all vertices with $d_i \leq K$.

3.4.3 Method

The whole method for solving (SPF) goes as follows.

First, we compute a good feasible solution using a memetic approach, see Section 3.8. This enables to find a first upper bound value on the problem.

Second, we solve the relaxation of (SPF) with the help of a column generation method, the set $\mathcal{R}$ being exponential, see Section 3.5. The way to compute the columns follows a scheme developed by Baldacci *et al.* [11]. It provides a lower bound. This iterative method is called several times. At each time, at most 200 negative reduce cost columns are added.

Third, the memetic algorithm is used again using the rounded up solution obtained by solving the relaxation of (SPF). This second call improves the upper bound found at the first step.

Fourth, still following Baldacci *et al.* [11], all columns that may play a role in an optimal solution, "columns in the gap", are added and the MIP is solved with the help of CPLEX. The fact that a column may play a role in an optimal solution can be checked with its reduced cost. Let g be the gap between the upper and lower bounds obtained so far. A solution containing a column whose reduced cost is strictly greater than g has a cost strictly greater than the upper bound. Columns in the gap are added and the MIP is solved. If all the columns in the gap have been added, the solution of the MIP is optimal (see discussion in Section 3.5).

3.5 Relaxation

The linear relaxation of (SPF) where constraints (3.4) are relaxed gives a lower bound on the value of the original problem. We obtain the following linear problem $(RSPF)$.

$$z(RSPF) = \min \sum_{r \in \mathcal{R}} c_r z_r \tag{3.5}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} b_{ir} z_r \geq d_i \quad \forall i \in V \tag{3.6}$$

$$\sum_{r \in \mathcal{R}} z_r \leq M \tag{3.7}$$

$$z_r \geq 0 \qquad \forall r \in \mathcal{R} \tag{3.8}$$

Note that in $(RSPF)$, constraints (3.6) are inequalities whereas in (SPF) constraints (3.2) are equalities. This operation enables to obtain signed dual variables and is done for stabilization of the algorithm. It may weaken the quality of the relaxation. However, in comparison with a more stable algorithm, it is worthwhile, and experiments shows that in general, it does not change the value of the relaxation, see Section 3.9.

As the set of routes $\mathcal{R}$ is exponential in the number of stations and bikes to move, we work with a column generation method. For $\overline{\mathcal{R}} \subseteq \mathcal{R}$, we define $(\overline{RSPF})$.

$$z(\overline{RSPF}) = \min \sum_{r \in \overline{\mathcal{R}}} c_r z_r \tag{3.9}$$

$$\text{s.t.} \sum_{r \in \overline{\mathcal{R}}} b_{ir} z_r \geq d_i \quad \forall i \in V \tag{3.10}$$

$$\sum_{r \in \overline{\mathcal{R}}} z_r \leq M \tag{3.11}$$

$$z_r \geq 0 \qquad \forall r \in \overline{\mathcal{R}} \tag{3.12}$$

The dual program (D) of $(\overline{RSPF})$ is the following one.

$$z(D) = \max \sum_{i \in V} \lambda_i d_i - \nu M \tag{3.13}$$

$$\text{s.t.} \sum_{i \in V} \lambda_i b_{ir} + \nu \leq c_r \quad \forall r \in \overline{\mathcal{R}} \tag{3.14}$$

$$\lambda_i \in \mathbb{R}_+, \nu \in \mathbb{R}_- \qquad \forall i \in V \tag{3.15}$$

$\lambda_i \in \mathbb{R}_+$ (resp. $\nu \in \mathbb{R}_-$) are the dual variables associated to constraints (3.10) (resp. (3.11)).

Columns generation method aims at adding only routes whose *reduced cost* $\overline{c_r} = c_r - \sum_{i \in V} \lambda_i b_{ir} - \nu$ is negative. Indeed, according to the theory of linear programming, these are the only routes that may improve the value of the objective function.

The *pricing subproblem* consists, given $\overline{\mathcal{R}}$, $\boldsymbol{\lambda}$ and $\rho \in \mathbb{R}$, in finding a route r such that

$$c_r - \sum_{i \in V} \lambda_i b_{ir} - \nu \leq \rho \quad \text{and} \quad r \in \mathcal{R} \setminus \overline{\mathcal{R}}. \tag{3.16}$$

To keep full generality, we have replaced the right hand side by an arbitrary real number ρ. When computing the lower bound, we will set $\rho := 0$. As outlined in Section 3.4.3, once a lower and an upper bounds are computed, we get the gap g between upper and lower bounds and we add all columns that may play a role in an optimal solution, in order to find this latter. Such columns have positive reduced costs less than g, and the method we present now is able to deal with such a case too by setting $\rho := g$.

The way to proceed to solve this subproblem is explained in the following section.

Remark Since the reduced costs are computed according to $(RSPF)$ and not with the most natural relaxation of (SPF) where (3.2) would still be an equality, it might happen that some routes playing a role in the optimal solution of (SPF) are missed. Nevertheless, in practice, such a situation never happens. It is consistent with the fact already noticed that the solutions of the relaxations experimentally coincide (and therefore the dual solutions and the reduced costs).

3.6 Solving the pricing subproblem

Pricing subproblem (3.16) can be expressed as a kind of Elementary Shortest Path with Resource Constraints (see for example [51]). The resource is here understood as the number of arcs traversed. We follow a method inspired by the one of Baldacci *et al.* [10]. They propose a column generation method to solve the PDPTW. Their procedure for finding negative reduce cost routes is divided into two phases.

First, *half routes* are computed. These half routes are either *forward paths* or *backward paths*: forward paths are directed paths starting at the depot and ending at a vertex after having

visited a subset of vertices; backward paths are directed paths starting at a vertex and visiting a subset of vertices before ending at the depot. This computation is done with a procedure called GENPATH.

Second, forward and backward paths are combined, while respecting compatibility constraints. Compatibility refers to the respect of precedence constraints or capacity for instance. This combination is done with a procedure called GENROUTE.

We follow this GENPATH-GENROUTE scheme. This method enables to consider all the routes and generate only the ones that may improve the value of the solution, identified by their negative reduce cost, if any exists. If this method fails to find any such a route, we know that there is no negative reduce cost route and we have solved the relaxation.

Forward and backward paths are also introduced but modified in order to fit the MVBP features. The main additional requirement is that the number of bikes to be loaded or unloaded at each stop has to be decided: vertices may be visited several times and pickup and delivery operations are not paired.

A *forward path* (resp. *backward path*) P is a sequence

$$s_P = (s_{1P}, s_{2P}, \ldots, s_{tP})$$

for some $t < \lfloor \frac{Rmax}{2} \rfloor$ (resp. $t < \lceil \frac{Rmax}{2} \rceil$) of visited vertices and a sequence

$$l_P = (l_{1P}, l_{2P}, \ldots, l_{tP})$$

of numbers of loaded or unloaded bikes at each stop. A forward path starts at the depot, $s_{1P} = 0$, and ends at a vertex denoted $e(P) \in V$, with a load on the vehicle denoted $\ell(P) \leq K$. A backward path ends at the depot, $s_{kP} = 0$, and starts at a vertex denoted $e(P) \in V$, with a load on the vehicle denoted $\ell(P) \leq K$. The cost of a path P is

$$\overline{c}_P = \sum_{k=1}^{t} (c_{s_{kP}s_{k+1P}} - \lambda_{s_{kP}} l_{s_{kP}}) - \frac{\nu}{2}$$

As for the routes, we define b_{iP} to be the number of bikes loaded or unloaded following P on a vertex i.

We define $\overrightarrow{\mathcal{P}}$ (resp. $\overleftarrow{\mathcal{P}}$) as the set of all such forward paths (resp. backward paths), with cost equal to or less than ρ and that satisfy the dominance rules of Section 3.4.1. These sets are generated in the GENPATH algorithm described hereafter (see Section 3.6.1), and they are used afterwards in the GENROUTE algorithm (see Section 3.6.2) to build routes by matching *compatible* forward and backward paths.

A forward path P and a backward path P' are said to be *compatible* if $e(P) = e(P')$, $\ell(P) = \ell(P')$, $b_{iP} + b_{iP'} \leq d_i$ for all $i \in V$, $|P| \leq \lfloor \frac{Rmax}{2} \rfloor$ and $|P'| \leq \lceil \frac{Rmax}{2} \rceil$, and the route obtained by gluing them together on $e(P) = e(P')$ satisfies the dominance rules.

Note that in an optimal solution, the first (resp. last) vertex to be visited by a route has to be a pickup one (resp. a delivery one). This remark is used in GENPATH.

3.6.1 The GENPATH procedure

The algorithm consists in an exploration of the set of possible paths, while using dominance rules and lower bounds to avoid the systematic enumeration of all paths. We explain it for forward paths, the adaptation for the backward case being straightforward.

Let P and $\tilde{P}$ be two forward paths with $|P|, |\tilde{P}| \leq \lfloor \frac{Rmax}{2} \rfloor$. We say that P *dominates* $\tilde{P}$ if $e(P) = e(\tilde{P})$, $\ell(P) = \ell(\tilde{P})$, $b_{iP} \leq b_{i\tilde{P}}$ for all $i \in V$ and $\overline{c}_P \leq \overline{c}_{\tilde{P}}$. The idea behind this definition is the following. Any backward path compatible with $\tilde{P}$ is also compatible with P. As the cost of P is smaller than the cost of $\tilde{P}$, discarding $\tilde{P}$ does not prevent of finding a negative reduced cost route, if there is one.

Three ideas are used in GENPATH. First, only paths satisfying the dominance rules of Section 3.4.1, and an additional dominance rule described hereafter in Section 3.6.3, are generated. Second, a lower bound $lb(P)$ on the cost of any backward path completing a forward path P is computed. If $\overline{c}_P + lb(P) > \rho$, path P is discarded. See Section 3.6.4. Third, if a forward path P is dominated by a path already in $\overrightarrow{\mathcal{P}}$, it is not added. If P is not dominated, then it is added to $\overrightarrow{\mathcal{P}}$ and all forward paths in this set dominated by P are deleted.

The procedure GENPATH is described in Algorithm 2.

Remark When the pricing subproblem is solved for finding routes with positive reduced costs ("columns in the gap", for finding the optimal solution of (SPF)), the notion of dominance described above is weakened and, instead, $b_{iP} = b_{i\tilde{P}}$ is required for all $i \in V$. Indeed, no column that may play a role in the optimal solution must be missed.

3.6.2 The GENROUTE procedure

We say that $r \in \mathbb{R}$ *dominates* $\widehat{r} \in \mathbb{R}$ if $b_{ir} = b_{i\widehat{r}}$ for all $i \in V$ and if $\overline{c}_r \leq \overline{c}_{\widehat{r}}$. There is obviously no interest to keep a route that is dominated by another one. The procedure GENROUTE is explained in detail in Algorithm 3. This method takes as an input a threshold ρ and

Algorithm 2 GENPATH for forward paths: return forward paths with cost $\leq \rho$

Step 1: Create the empty path P_0 with $\overline{c}(P_0) = 0, e(P_0) = 0, \mathcal{T} = \{P_0\}$ and $\overrightarrow{\mathcal{P}} = \emptyset$

Step 2:

if $\mathcal{T} = \emptyset$ **then**

STOP

end if

Step 3: $P^* = \arg\min\{lb(P), P \in \mathcal{T}\}$

$\mathcal{T} = \mathcal{T} \setminus \{P^*\}$

Insert in $\overrightarrow{\mathcal{P}}$ forward path P^*

if $|P^*| = \left\lfloor \frac{Rmax}{2} \right\rfloor$ **then**

Return to Step 3

end if

for $i \in V$ and $\delta \in \{-K, \ldots, K\}$ **do**

Expand P^* from $e(P^*)$ to vertex i with new load $\ell(P^*) + \delta$ to obtain $\widehat{P}$

$\overline{c}_{\widehat{P}} = \overline{c}_{P^*} + c_{e(P^*)i} - \lambda_i \delta$

Compute $lb(\widehat{P})$

if Dominance rules or capacity constraints not satisfied, $lb(\widehat{P}) > \rho$ or there is already a path that dominates $\widehat{P}$ **then**

Reject $\widehat{P}$

else

Remove all forward paths in $\mathcal{T}$ dominated by $\widehat{P}$

Insert $\widehat{P}$ in $\mathcal{T}$

end if

end for

Step 4: return to Step 2

all the forward paths and backward paths that were created by GENPATH. The set $\overrightarrow{\mathcal{P}}$ (resp. $\overleftarrow{\mathcal{P}}$) of forward (resp. backward) paths is assumed to be partitioned into sets $\overrightarrow{\mathcal{P}}_{iw}$ (resp. $\overleftarrow{\mathcal{P}}_{iw}$) of forward (resp. backward) paths P with $\ell(P) = w$ and $e(P) = i$.

Algorithm 3 GENROUTE procedure: return a set $\mathcal{R}$ of routes with cost $\leq \rho$

$\mathcal{R} = \emptyset$
for $i \in V$ and $w \in \{0, \ldots, K\}$ **do**
  for $P \in \overrightarrow{\mathcal{P}}_{iw}$ and $P' \in \overleftarrow{\mathcal{P}}_{iw}$ **do**
    $R \leftarrow P \cup P'$
    if R is a feasible and not dominated in $\mathcal{R}$ **then**
      Add R to $\mathcal{R}$
      Delete from $\mathcal{R}$ all routes dominated by R
    end if
  end for
end for

To increase the speed of the algorithm, $\overrightarrow{\mathcal{P}}_{iw}$ and $\overleftarrow{\mathcal{P}}_{iw}$ are sorted according to the costs, before running GENROUTE: one can make $i \leftarrow i + 1$ when P and P' are such that the sum of their costs is strictly greater than ρ.

3.6.3 Additional dominance rules in GENPATH

The following remark helps to cut routes while trying to find routes with cost strictly less than 0 (and not when $\rho \geq 0$). Note that these dominance rules use the dual values λ.

If there is a sequence visiting $s \rightarrow s_p \rightarrow s_d \rightarrow s'$ such that $s_p \in \mathcal{P}i$ a pickup vertex from where w bikes are loaded and $s_d \in \mathcal{D}e$ a delivery one where w' are unloaded, then a new solution with the same sequence but skipping either s_p either s_d or both of them could have a lower cost in the following cases:

if $(w < w')$ **and** $(c_{ss_d} + w(\lambda_{s_p} + \lambda_{s_d}) - (c_{ss_p} + c_{s_p s_d}) < 0)$ the forward path skipping s_p and unloading only $(w' - w)$ bikes at s_d has a lower cost

if $(w = w')$ **and** $(c_{ss'} + w(\lambda_{s_p} + \lambda_{s_d}) - (c_{ss_p} + c_{s_p s_d} + c_{s_d s'}) < 0)$ the forward path skipping both s_p and s_d has a lower cost

if $(w > w')$ **and** $(c_{s_p s'} + w'(\lambda_{s_p} + \lambda_{s_d}) - (c_{s_p s_d} + c_{s_d s'}) < 0)$ the forward path skipping s_d and loading only $(w - w')$ bikes at s_p has a lower cost

A similar remark leads to similar conclusions when the sequence is $s \to s_d \to s_p \to s'$, with $s_d \in \mathcal{D}e$ and $s_d \in \mathcal{P}i$.

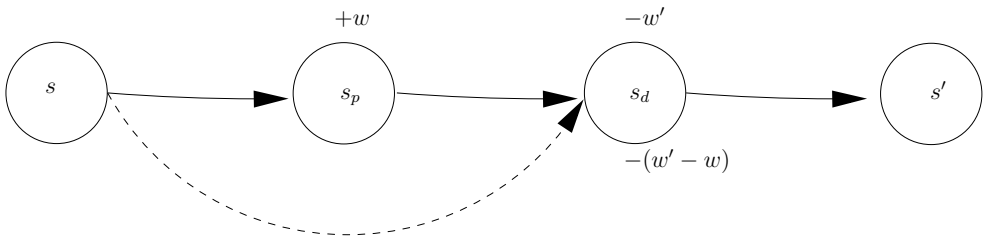

Figure 3.3: New forward path when $w < w'$ and when the cost of the forward path skipping s_p is lower than the one of the original

3.6.4 Lower bound lb for GENPATH

GENPATH procedure needs an idea of the cost that a forward path (respectively a backward path) would have if completed. For that purpose, a lower bound on the cost of returning to the depot from a given vertex with a given load on the vehicle (resp. going at a given vertex with a given load) is computed. A dynamic programming method to compute such a lower bound is exposed hereafter.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be an acyclic oriented graph where

$$\mathcal{V} = \{(i, w, a) : \, i \in V, w \in \{0, \ldots, K\}, \, a \in \{0, \ldots, Rmax - 1\}\} \, .$$

A vertex (i, w, a) represents a vehicle at station i with w bikes loaded on it after having traversed a arcs in G from the depot.

An arc in $\mathcal{G}$ links (i, w, a) to (i', w', a') when it encodes a possible transition, that is when

- $a' = a + 1$
- $i \neq i'$
- $w' - w \leq \min\{d_{i'}, K - w\}$ if $i' \in \mathcal{P}i$
- $w - w' \leq \min\{w, d_{i'}\}$ if $i' \in \mathcal{D}e$.

The cost of an arc linking (i, w, a) to (i', w', a') is defined as:

$$\widetilde{c}_{(i,w,a),(i',w',a')} = c_{ii'} - \lambda_{i'}|w' - w|.$$

The cost of shortest path $\overrightarrow{f}(i, w, a)$ reaching (i, w, a) from $(0, 0, 0)$ through dynamic programming.

$$\overrightarrow{f}(0,0,0)= 0 \tag{3.17}$$

$$\overrightarrow{f}(i,w,a) = \min_{(i',w',a-1)\in\mathcal{V} \text{ s.t. } ((i',w',a-1),(i,w,a))\in\mathcal{A}} \{\overrightarrow{f}(i',w',a-1) + \widetilde{c}_{(i',w',a-1),(i,w,a)}\} \tag{3.18}$$

This value is a strict lower bound because the history is not kept: there is no trace of the total number of bikes loaded or unloaded. It may happen that the number of bikes moved at some vertices while following this shortest path exceeds their demand.

Define $\overrightarrow{F}(i,w) = \min_{a\in\{1,\ldots,Rmax-1\}} \overrightarrow{f}(i,w,a)$. This value is the minimum of all the shortest paths going from the depot to a vertex i with w bikes loaded on the vehicle within strictly less than $Rmax$ arcs. It is a lower bound on the cost of any forward path going from the depot to i and finishing with w bikes.

A similar method using an acyclic graph leads to the calculus of $\overleftarrow{F}(i,w)$ that is the minimum of all the shortest paths going from vertex i with load w to the depot within strictly less than $Rmax$ arcs. It provides a lower bound on the cost of any backward path going from a vertex i to the depot and starting with w bikes.

Finally, the lower bound $lb(P)$ on the cost to complete a path P, used in Section 3.6.1, is

$$lb(P) := \begin{cases} \overleftarrow{F}(e(P),\ell(P)) & \text{if } P \text{ is a forward path.} \\ \overrightarrow{F}(e(P),\ell(P)) & \text{if } P \text{ is a backward path.} \end{cases}$$

3.7 Adding cuts to increase the $z(\overline{RSPF})$

Adding new constraints to the primal problem leads to new dual variables. Reduced cost of routes are modified accordingly. For each type of cuts, its implications on the reduced cost is briefly explained and their influences in the reduce cost of the route are taken into account in both procedures GENPATH and GENROUTE.

Two types of cuts are added. A first family of cuts – *dual-feasible function cuts* – is obtained through dual-feasible functions, some of them are taken from [26] and the other are special-purposed ones. A second family – *dominance cuts* – comes from the generalization of some of the dominance rules of Section 3.4.1 to the pool of routes.

3.7.1 Dual-feasible function cuts

Dual-feasible functions are added to enhance the model. Three different types have been added: the Fekete-Schepers functions $(f)^k_{FS,1}$ and the Carlier *et al.* function $(f)^k_{CCM,1}$, see

[22, 40, 26] and the other are special-purposed ones. Here we expose the special-purposed ones and how their influence in the reduce cost is taken into account.

Let $k \leq d$ be two positive integers. We define for $x \in [0, d]$ the map

$$F_{k,d}(x) := \begin{cases} 2j & \text{if } x \in \left(\frac{jd}{k}, \frac{(j+1)d}{k}\right) \\ 2j-1 & \text{if } x = \frac{jd}{k} \end{cases} \qquad \text{where } j = \{0, \ldots, k-1\}$$

Proposition 3.7.1 (Superadditivity of $F_{k,d}$)**.** *For all $x, y \in [0, d]$ such that $x + y \leq d$, we have*

$$F_{k,d}(x) + F_{k,d}(y) \leq F_{k,d}(x + y).$$

Proof. Three cases have to be checked. For each of them, the inequality is straightforward.

- $x = \frac{j_x d}{k}$ and $y = \frac{j_y d}{k}$: indeed $2j_x - 1 + 2j_y - 1 \leq 2(j_x + j_y) - 1$.
- $x \in \left(\frac{j_x d}{k}, \frac{(j_x+1)d}{k}\right)$ and $y = \frac{j_y d}{k}$: indeed, $2j_x + 2j_y - 1 \leq 2(j_x + j_y)$.
- $x \in \left(\frac{j_x d}{k}, \frac{(j_x+1)d}{k}\right)$ and $y \in \left(\frac{j_y d}{k}, \frac{(j_y+1)d}{k}\right)$: indeed, $2j_x + 2j_y \leq 2(j_x + j_y)$

□

For $k \leq d_i$, we apply F_{k,d_i} on $\sum_{r \in R} b_{ir} z_r = d_i$. With the supperadditivity of F_{k,d_i}, we get that

$$\sum_{r \in R} F_{k,d_i}(b_{ir}) z_r \leq F_{k,d_i}(d_i)$$

is a valid inequality. Denote $\mu_{k,i} \leq 0$ the dual variable associated to this inequality. The pricing subproblem (3.16) becomes

$$c_r - \sum_{i \in V} (\lambda_i b_{ir} + \mu_{k,i} F(b_{ir})) \leq \rho \quad r \in \mathcal{R} \setminus \overline{\mathcal{R}}. \tag{3.19}$$

GENPATH can be adapted to deal with these new constraints. For forward paths, we will get something like

$$\overline{c}_P = c_P - \sum_{i \in V} (\lambda_i b_i(P) + \mu_{k,i} F_{k,d_i}(b_i(P))).$$

And for backward paths

$$\overline{c}_P = c_P - \sum_{i \in V} (\lambda_i b_i(P) + \mu_{k,i} F_{k,d_i}(b_i(P))).$$

Using the superadditivity of F_{k,d_i}, we have

$$\overline{c_P} + \overline{c_P} = c_r - \sum_{i \in V} (\lambda_i b_{ir} + \mu_{k,i}(F_{k,d_i}(b_i(P)) + F_{k,d_i}(b_i(P)))) \leq c_r - \sum_{i \in V} (\lambda_i b_{ir} + \mu_{k,i} F_{k,d_i}(b_{ir})).$$

GENPATH won't miss routes whose reduced cost is under ρ. The computation of the lower bound lb can be similarly straightforwardly adapted again because of the superadditivity of the F_{k,d_i}'s. In practice, see Section 3.9, the maps $F_{k,d}$ are used for $k = 2$ and $k = 3$.

3.7.2 Dominances cuts

The second type of cuts are obtained thanks to the dominances formerly defined in Section 3.4.1. Proposition 3.4.1 implies that an edge connecting two pickup vertices or two delivery vertices is used in at most one route. It provides "clique-type" constraints of the following form

$$\sum_{r \in R^1_{i,j}} z_r \quad \leq \quad 1 \qquad i,j \text{ vertices of same type} \tag{3.20}$$

where $R^1_{i,j}$ is the set of all feasible routes containing the arc (i,j) or the arc (j,i).

Proposition 3.4.2 implies similar constraints of the following form

$$\sum_{r \in R^2_{i,j}} z_r \quad \leq \quad 1 \qquad i,j \text{ vertices of distinct type} \tag{3.21}$$

where $R^2_{i,j}$ is the set of all feasible routes containing the arc (i,j) with a load strictly less than K or strictly greater than 0 according to the type of vertices i and j.

These constraints can easily be checked while building the paths and the routes. However, they can not be taken into account in the lower bound lb (dynamic programming has no memory). It is not too problematic since the lower bound computed without them remains valid.

3.8 Upper bound

To obtain an upper bound on the value of the solution, a Memetic Algorithm (MA) is used. MA have been introduce by Moscato [62]. As in the genetic algorithm, a population of individuals is kept. A score is associated to each individual. The objective is to find the minimal cost individual. For that purpose, the population is subject to several transformations: individuals are crossed together to create new individuals; local searches are performed on individuals to have a population of local minima. To define the MA, we have to describe the individuals and its score, the cross-over operations, the local searches and the management of the population.

3.8.1 Individuals

In the one-vehicle version of the balancing problem, a polynomial algorithm has been proposed to check whether there are bike displacements compatible with a given sequence of vertices that bring the system to its target state (see Section 2.3 in the former chapter). The algorithm uses a flow representation. Such an algorithm is useful since it allows to work only with

sequences of vertices when exploring the solution space, instead of working with sequences of vertices and bike displacements between the vertices. In this section, we show that such an algorithm exists also for the case with many vehicles. Note that in the SVOCPDP, drops were allowed, which is not the case in the MVBP. At first glance, it seems that the flow algorithm cannot be adapted. However, a simple trick provides a way to forbid drop in the flow modelling.

Proposition 3.8.1. *Let $0 = i_1^\mu, i_2^\mu \dots, i_{t_\mu}^\mu = 0$ be a collection of sequences of vertices for $\mu = 1, \dots, M$, the integer t_μ being the length of the μth sequence. There is a polynomial algorithm finding new initial and target states (p'_i, q'_i) for each vertex i and M routes inducing these sequences of vertices and bringing the system from (p'_i) to (q'_i) such that*

- *$0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i*
- *$\sum_i p'_i = \sum_i q'_i$*
- *the quantity $\sum_{i \in V} p'_i$ is maximal.*

In particular, it is possible to decide in polynomial time whether a collection of M sequences of vertices is induced by M routes at the end of which the system is at its target state (in this later case, $p'_i = p_i$ for each vertex i).

Proof of Proposition 3.8.1. For each $\mu = 1, \dots, M$, we build an oriented graph $D^\mu = (U^\mu, A^\mu)$ as follows. U^μ has t_μ elements: each vertex i_k^μ (make as many copies of a vertex i of G as there are occurrences of i in the sequence). The arcs in A^μ are of two types:

1. one arc (i_k^μ, i_{k+1}^μ) for each $k = 1, \dots, t_\mu - 1$, with capacity K, and
2. one arc between the bth occurrence of each vertex i^μ and its $(b+1)$th occurrence, if there is one, with capacity $+\infty$,

This graph D^μ is more or less the one used in the case with one vehicle of Figure 2.6 of the former chapter. Now, we put an arc between the last occurrence of a vertex in D^μ with its first occurrence in $D^{\mu+1}$ (or $D^{\mu+2}$ etc. – choose the first $D^{\mu'}$ with $\mu' > \mu$ with an occurrence of this vertex if there is one).

Finally, we add two special vertices s et t (the source and the sink). There is an arc from s to each first occurrence of a pickup vertex in $\bigcup_{\mu=1}^M D^\mu$ and there is an arc from the last occurrence of each delivery vertex to t. Their capacities are respectively p_i and q_i.

Denote by D this graph. Note that it has each of the D^μ has a subgraph.

Any s-t flow on D encodes possible bike displacements compatible with the given sequences of vertices. The numbers of bikes to be moved by vehicle μ while going from i_k^μ to i_{k+1}^μ are given by the flow on arc (i_k^μ, i_{k+1}^μ) (arcs defined in 1.); the number of bikes remaining

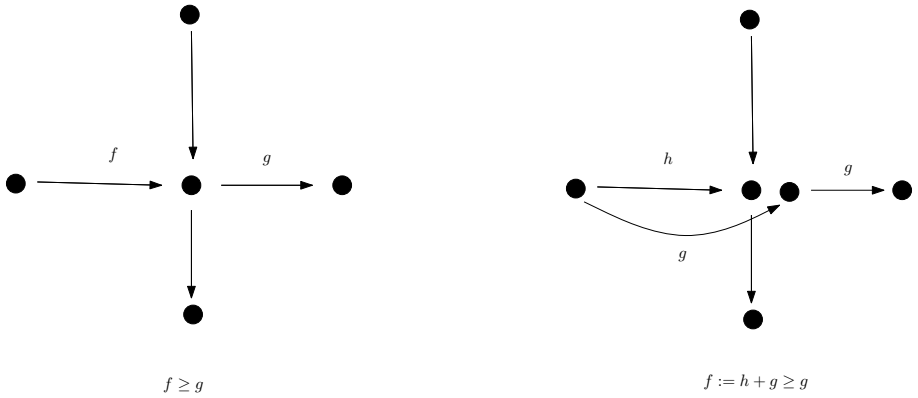

Figure 3.4: How to impose an inequality between the entering and the leaving flows in a vertex

in a vertex i after the bth visit of the vehicle μ is given by the flow on the arc between the bth occurrence of the vertex i and the $(b+1)$th occurrence (arcs defined in 2. or arcs between a D^μ and a $D^{\mu'}$ with $\mu' > \mu$ if there is no next occurrence of this vertex for vehicle μ).

The initial number of bikes in a vertex i is given by the flow on the arc between s and the first occurrence of the vertex in D. The final number of bikes in a vertex i is given by the flow on the arc between its last occurrence and t.

And conversely, any bike displacements compatible with the given sequences of vertices induce an s-t flow.

Now, there is still a constraint missing: we must forbid drops. It means that between the flow f on arc (i^μ_j, i^μ_{j+1}) and the flow g on arc $(i^\mu_{j+1}, i^\mu_{j+2})$, as defined in 1., there must be an inequality: $f \leq g$ if i^μ_{j+1} is a pickup vertex and $f \geq g$ if it is a delivery vertex. There is an easy way for forcing this kind of inequality in a flow problem, by changing the origin of an arc and adding a new vertex (see Figure 3.4 for which one requires $f \geq g$).

p'_i is then the value of the flow in the arc between s and the first occurrence of i, and q'_i the value of the flow in the arc between the last occurrence of i and t. If a vertex i is not present in any sequences, then we set $p'_i = q'_i = \min(p_i, q_i)$. □

This result enables to consider only sequences of vertices, forgetting the logistic operations to be done at each vertex. For a given sequence, the max-flow algorithm used in the proof of Proposition 3.8.1 allows to recover the number of bikes to be loaded or unloaded at each vertex.

The individual is encoded as a $M \times Rmax$ matrix, each line referring to a distinct route.

3.8.2 Score of an individual

The score of an individual is the total length of all its routes plus a penalty proportional to the number of bikes that are misplaced at the end of them. The number of misplaced bikes is computed with the help of Proposition 3.8.1.

3.8.3 Cross-Over Operation

To perform the cross-over operations, we adapt the route-first, cluster-second method of Prins [73]. To that purpose, a giant route is defined as the sequence of routes sticked together without trip delimiters. When performing the cross-over, two individuals are selected and transformed into giant routes. The giant routes are recombined together as follows: a cut is drawn at random; the first part of the first giant route is selected and completed by the second part of the second giant route, and vice-versa. To be sure to visit all vertices at leasts once, if some of the vertices are not visited once the recombination is done, they are introduced at the cut with respect to their order in the original giant routes.

Once a new giant route is created, the original one-vehicle version [24] of Proposition 3.8.1 is used as a preliminary test: If the new giant route is not feasible for the one-vehicle version, no division of the giant routes into M routes gives a feasible solution for the MVBP and the new giant route can be discarded. However, if it is feasible for the one-vehicle version, it does not necessary imply the feasibility for the MVBP.

In the case the preliminary test is successful, the new giant route is divided with the procedure of Prins [73] using an acyclic directed graph $\mathcal{D}$. The vertices of $\mathcal{D}$ are the vertices of the giant route (with repetitions). An arc is created between two vertices if there are at most $(Rmax - 3)$ vertices between them on the giant route. In this way, the sequence between the two endpoints of such an arc can lead to a route for a vehicle. Moreover, the following conditions are also required to create an arc (u, v) as they are *natural conditions* for an optimal route.

- u is a pickup vertex.
- v is a delivery vertex.
- The depot is not between u and v in the giant route.
- There are no consecutive occurences of the same vertex between u and v in the giant route.

An arc (u, v) of $\mathcal{D}$ gets as a weight the length of the closed path starting and ending with the

depot and going through the sequence of vertices between u and v on the giant route. The division is obtained thank dynamic programming: we look for a directed path in $\mathcal{D}$ of minimal weight, using at most M arcs. The minimization is over the sum of the lengths, which is not the exact objective as we aim at balancing the network. However, the local searches presented in the following paragraph are here to compensate this effect.

3.8.4 Local searches

Three local searches are performed on the individuals in order to find local minima. The first one is a 2-OPT within a route – line of the matrix representing the individual. The second is a 2-OPT between two routes – two lines of the matrix.

A third local search is tried when the individual is not feasible in the sense of Proposition 3.8.1 aiming at decreasing the penalty (see Section3.8.2). Stations are browsed to identify the unbalanced vertices. Then, if a route traverses $Rmax$ arcs, it tries to add unbalanced vertices as follows:

- If any unbalanced pickup vertex is visited, it tries to add any delivery vertex after this visit
- If any unbalanced delivery vertex is visited, it tries to add any pickup vertex before this visit
- Otherwise, if two new stops can be added, it tries to add an unbalanced pickup vertex and an unbalanced delivery vertex at any point of the route, the former being before the latter.

Only the move that provides the best improvement is done. After a move, a cleaning operation is performed. It enables to ensure that the natural conditions for a route are satisfied (see the conditions of existence of arc (u, v) in the previous subsection).

3.8.5 Population and selection

At each iteration, the new individuals are scored and if their scores are lower than the one of the worst individuals, they are inserted into the population while the latter are erased. The size of the population is kept constant over time.

The selection of the individuals for the crossing-over is not uniform over all the population: the first one is chosen with a greater probability on the lowest score individuals, the second individual probability is uniform over all the population without the first individual.

3.8.6 MA calls in the overall algorithm

This MA is used twice in the overall algorithm (see Section 3.4.3). For the first call, two individuals are created using greedy heuristics. The first one is to use the solution of the SVOCPDP as a giant route (see former Section 3.8.1). The second uses a method similar to the one used in the SVOCPDP and explained in Section 2.8.2. The others are randomly generated. This call is used to build the first set $\overline{\mathcal{R}}$ of feasible routes.

The second call is performed after having computed a lower bound, in order to get a good upper bound. For that call, the population is enriched by a solution obtained by rounding up some of the z_r solution of the relaxation and solving it again until all z_r are integers.

3.9 Computational study

All the algorithms have been coded in C++ and tested on a PC AMD Athlon 5600+ clocked at 2.8 GHz, with 16 GB RAM. CPlex 12 is used.

3.9.1 Instances and results

The column generation algorithm runs for 6 hours. M is set to 5 and $Rmax$ to 10. The time limit on the first call of the MA was set to 300 seconds. The algorithm was tested on two sets of instances.

The first set of instances are those used for the SVOCPDP, completed with other instances with 10 stations. They latter were created from the 20 stations instances of the SVOCPDP split into two parts. The vehicles capacity K is set to 10 in all the instances. Table 3.1 shows the results obtained on instances with up to 30 stations. Experiments have been done for $d_i \leq K$. The name of an instance is of the form "nαqβX" where α is the number of stations and β is the capacity of the vehicle. It is shown in the first column. The second column gives the number of non initially balanced vertices. The third columns presents $\overline{d} = \sum_{i \in V} d_i$ which is twice the number of bikes to move. Columns LB and LBc show the lower bound obtained, respectively before and after having added the cuts of Section 3.7. These cuts are not initially added in the linear program. At first, the linear problem $(\overline{RSPF})$ is solved to optimality without any cuts added. Then we enter a second phase during which we follow the same framework but we can add at each iteration columns and cuts, if any violated cut is found. The sixth column

LB_{SV} gives the lower bound obtained solving the SVOCPDP, which is also a relaxation of the multiple vehicle problem. The number of dual-feasible function cuts (FC) and of clique cuts (CC) used for computing LBc are shown in the two following columns.

The final upper bound (UB) obtained when the algorithm stops after a time limit is given in the nineth column. A star $*$ is added when the UB is in fact the optimal value. The fact that it is the optimal value is proven *a posteriori* by checking that the values of the relaxations – the one with the equality as in (3.2) and the one with the inequalities as in (3.6) – are the same (see discussion in Section 3.5). The gap presented in the tenth column is computed using the former value and the best lower bound obtained, in the case LBc and LB_{SV} exist. Otherwise, LB_{SV} is used. UB_{MA} gives the value obtained after the first call to the memetic algorithm. The two last columns give the time spent by the whole method and the final number of columns.

The MA has moreover been experimented on the 30 and 40 stations instances of this first set. Table 3.2 shows the results on these instances when the MA is run for 2 hours instead of 300 seconds. The first column gives the name of the instance, the second columns $LB^{\#}$ gives the best lower bound obtained. For the 30 stations instances, it is the higher value between LBc and LB_{SV} when the latter is found in Table3.1. For the 40 stations instances, it is the lower bound obtained for the SVOCPDP. The third column $UB^{\#}_{MA}$ shows the final value obtained at the end of the memetic call. The last column is the gap between the two former values.

The second set of instances are the SDVRP instances taken from [9] that were modified as follows: for these nine instances, the capacity of the vehicles was set to 100 and the demand of the station are either 90 or 60. All the former numbers are divided by 10 since it does not change the optimal solution on the instances. The bound $Rmax$ can then be 10 as it is the capacity of the vehicle: no route can be longer than the vehicle capacity. For these instances, M is set to infinity since in the SDVRP the number of vehicles is unbounded and therefore the constraints (3.3), (3.7) and (3.11) are forgotten. To match with the MVBP, a pickup vertex at a zero-distance from the depot receives all the commodities to be delivered at all the other vertices.

Table 3.3 shows the results obtained running the method explained here on these instances. The first column shows the name of the instance. The second gives the number of unbalanced vertices in the instances. LBc shows the lower bound obtained after dual-feasible function cuts and clique cuts are added. UB is the upper bound obtain at the end of the method and the column "Gap" informs on the gap between UB and the best lower bound obtained. Here again

a star $*$ is added when the UB is the optimal value. Column "Time" presents the time spent by the method. The two last columns give respectively the lower bound and upper bound obtained on these instances in [9]. When these two values are equal, the value is then the one of the optimal solution, and it is marked with a star $*$.

3.9.2 Conclusion

The optimal solutions are found for all the instances with 10 vertices and in very short time. For the instances with 20 stations, the optimal solution is found in some cases. In the other cases, the gap is quite small, except in one case, the instance n20q10H. Note that there can be an irreducible gap between the lower bound computed from the relaxation and the optimal solution, as it appears in the 10 stations instances. This gap would be the gap at the root node if any branching were performed. For the 30 stations instances, the gap increases a lot. In some cases, the lower bounds LB or LBc are not found, in which case the mark "-" is written in the corresponding cell of Table 3.1.

The MA enables to find very good solution in the 10 stations instances, as in most of the case UB_{MA} is the optimal solution. In 6 cases out of 20, it is near the optimal solution. In the 20 stations instances, UB_{MA} and UB are very close except in one case, the same instance as before. When the optimal solution is found at the end, it was already found by the memetic algorithm. As for the 30 stations instances, the method does not improve the solution obtained by the MA in some case, because of the time limit. The gap is then more significant. Table 3.2 shows the result of the memetic algorithm alone. Run with more time, the memetic algorithm improves the quality of the solution: for all the 30 stations instances, $\text{UB}^{\#}_{MA} \leq \text{UB}_{MA}$. For the 40 stations instances, the gap is bigger. Note that the only available lower bound is the one of the SVOCPDP. This bound could be quite loose when the number of stations increases. Indeed the distance between LBc and LB_{SV} is smaller in the 10 stations instances than in the 30 station ones.

For the SDVRP instances, Table 3.3 shows that 5 out of the 6 optimal solutions found by Archetti *et al.* [9] are obtained with the method exposed in this chapter. In the other cases, the upper bound is equal or near to the value already found. The lower bound is much lower than the one found by Archetti *et al.* [9]. This is expected as the MVBP is more general. Special cuts suitable for the SDVRP may be used to enhance this value.

The computation time and the size of the problem solved do not enable to plan to use this algorithm for solving the MVBP at a city scale: even small BSS have a few dozens of stations, and this number can exceed $1'400$ as in Paris. However, in some systems, the number of stations requiring at least one visit per day is much smaller: for instance, in the Brussel Villo! system, there are about forty of them [21] while the whole system has 180 stations. The method outlined in this chapter could then be used for solving a reduced problem on the subset of problematic stations.

Instance	$\|\mathcal{P}i\| + \|\mathcal{D}e\|$	$\overline{d}$	LB	LBc	LB_{SV}	# FC	# CC	UB	Gap %	Time	UB_{MA}	$\|\overline{\mathcal{R}}\|$
n10q10A	10	52	3049.21	3055.00	2994	0	1	3055*	0.00	188.93	3130	955
n10q10a	10	48	3572.09	3611.25	3483	2	6	3719*	2.98	294.35	3719	1273
n10q10B	10	60	3619.95	3631.95	3671	1	5	3704*	0.89	342.18	3704	2629
n10q10b	10	32	3026.00	3114.50	3122	5	4	3192*	2.24	248.44	3192	1072
n10q10C	10	64	3133.38	3258.00	3316	4	16	3392*	2.29	2441.71	3392	6556
n10q10c	10	44	4239.00	4239.00	4226	0	0	4239*	0.00	191.43	4239	929
n10q10D	10	54	3109.18	3147.33	3071	0	6	3199*	1.64	1205.95	3273	1834
n10q10d	10	42	4002.05	4206.83	4238	4	9	4497*	6.11	359.88	4497	17523
n10q10E	10	58	4680.66	4857.29	4828	1	8	4876*	0.38	469.50	4921	5589
n10q10e	10	58	3627.29	3675.60	3816	2	10	3823*	0.18	948.44	3823	23537
n10q10F	10	50	3685.60	3794.88	3758	1	7	3796*	0.02	226.16	3836	9297
n10q10f	10	36	3208.44	3283.46	2908	0	6	3468*	5.62	276.07	3468	950
n10q10G	10	52	3713.89	3735.61	3325	2	8	3973*	6.35	716.29	4151	2445
n10q10g	10	34	4020.38	4075.20	3685	0	5	4179*	2.54	285.94	4179	1084
n10q10H	10	48	3710.95	3772.06	3790	2	7	3959*	4.45	1705.69	3959	4265
n10q10h	10	62	3995.12	4043.89	3960	2	8	4168*	3.06	5292.60	4168	4713
n10q10I	10	44	3604.02	3711.20	3138	1	10	3963*	6.78	361.36	4026	1802
n10q10i	10	60	2614.47	2632.50	2390	1	3	2645*	0.47	9663.33	2645	824
n10q10J	9	56	3117.00	3125.00	3060	0	1	3125*	0.00	909.75	3125	2488
n10q10j	10	42	3329.06	3392.40	3139	3	9	3453*	1.78	223.70	3453	2099
n20q10A	17	88	4679.25	4758.00	4702	4	12	4826*	1.42	2547.20	4826	3917
n20q10B	18	80	4994.09	5051.63	4769	3	10	5460	8.08	34244.00	5751	337467
n20q10C	20	108	6214.24	6317.00	6012	3	16	6509	3.03	22747.00	6723	138115
n20q10D	19	118	6136.87	6208.00	5989	7	8	6208*	0.00	1122.67	6208	3723
n20q10E	18	112	6124.96	6217.29	6245	6	17	6520	4.40	22209.00	6535	303986
n20q10F	20	90	5085.13	5162.82	4717	6	15	5222*	1.14	1610.25	5222	3506
n20q10G	19	86	5437.96	5466.29	5070	7	13	5795	6.01	21912.50	5795	17679
n20q10H	18	100	5643.31	5824.36	5542	5	24	6582	13.00	161890.00	6583	115217
n20q10I	19	102	4788.49	4879.42	4576	0	20	5225	7.08	23419.00	5333	249917
n20q10J	16	92	4476.65	4545.00	4070	12	9	4545*	0.00	2154.61	4545	3360
n30q10A	29	140	6527.70	6637.53	6236	6	20	7265	9.45	23803.00	7712	204394
n30q10B	25	124	6528.49	6676.48	6308	8	22	7041	5.45	23370.10	8081	25938
n30q10C	28	152	-	-	6335	-	-	8050	27.07	21600.00	8050	-
n30q10D	28	146	6596.43	6729.89	6076	5	26	8127	20.75	36449.20	8127	6519
n30q10E	26	132	6363.72	6497.71	5877	5	22	7507	15.53	22099.00	7532	205270
n30q10F	29	130	6329.06	6450.86	5695	4	20	7335	13.70	22990.00	7335	110319
n30q10G	27	162	8721.26	8879.59	8891	7	28	10011	12.59	38467.00	10011	114893
n30q10H	27	136	6431.10	6640.81	5884	6	27	7464	12.39	21906.00	7464	94038
n30q10I	27	144	5760.94	5974.50	5430	4	31	6923	15.87	50727.40	6923	3371
n30q10J	27	138	-	-	5764	-	-	7672	33.10	21600.00	7672	-

Table 3.1: Results on the instances with up to 30 stations and $d_i \leq 10$

Instance	LB$^{\#}$	UB$^{\#}_{MA}$	gap %
n30q10A	6637.53	7166	7.96
n30q10B	6676.48	6939	3.93
n30q10C	6335.00	7575	19.57
n30q10D	6729.89	7543	12.08
n30q10E	6497.71	7209	10.94
n30q10F	6450.86	6715	4.09
n30q10G	8891.00	9693	9.02
n30q10H	6640.81	7069	6.44
n30q10I	5974.50	6339	6.10
n30q10J	5764.00	6953	20.62
n40q10A	6711.66	9675	44.15
n40q10B	5949.00	9012	51.48
n40q10C	7237.00	8258	14.10
n40q10D	7692.00	9488	23.34
n40q10E	6424.00	8595	33.79
n40q10F	6651.54	9552	43.60
n40q10G	7384.00	9933	34.52
n40q10H	6556.00	9339	42.44
n40q10I	6901.00	9540	38.24
n40q10J	5989.20	8297	38.53

Table 3.2: Results of the memetic on the instances with up 30 and 40 stations

Inst.	n	LBc	UB	gap %	time (s)	$\underline{z}^*$	$\overline{z}^*$
SD1	9	21555.60	22828*	5.90	52	22828*	22828*
SD2	17	66476.00	70828*	6.54	1018	70828*	70828*
SD3	17	42844.00	43044*	0.46	249	43044*	43044*
SD4	25	62895.60	63062*	0.26	400	63062*	63062*
SD5	33	132515.00	138994	4.88	20229	137325	138994
SD6	33	82988.90	83086*	0.11	1717	83086*	83086*
SD7	41	345261.00	366000	0.06	21600	363844	364000
SD8	49	486218.00	514046	5.72	28115	506828*	506828*
SD9	49	197319.00	206519	4.66	17767	202808	204288

Table 3.3: Results on the SDVRP instances with up to 50 stations

Part II

The dynamic problem

Chapters 4 and 5 gather works done in collaboration with Frédéric Meunier, Roberto Wolfler Calvo, Houssame Yahiaoui and Thomas Pradeau. Houssame Yahiaoui implemented the simulator presented in Chapter 4 and I helped to build its model. Thomas Pradeau and Frédéric Meunier initiated the work on the complexity study of Chapter 5 which I completed. I worked on the real time regulation algorithms. This work was presented at the INFORMS 2011 and ROADEF 2012 conferences, and a journal version of this work is in preparation [25].

Chapter 6 presents results of a common work in collaboration with Tal Raviv and Michal Tzur. A journal version of this chapter is in preparation.

Chapter 4

Real-time shared transport system: model and simulations

4.1 Introduction

Shared transport systems gather different means of transportation with common features. The idea is to share vehicles between several users. In car sharing and bike sharing systems, the vehicles are not owned by users. A fleet of vehicles are available. As stated in Chapter 1, the third generation of these transportation solutions is the focus of this work. In this case, cities are equipped with stations. The number of stations depends on the size of the city. In the BSSs existing already, this number goes from few stations to several thousands of stations in the extreme cases. The stations are located in order to cover all the city, preventing imbalances in catchment areas. Each station has a capacity in vehicles, depending on the number of parking places installed. For instance in Paris there is a Vélib' station every 300 meters on average and the number of bike racks per station goes from a dozen to over sixty. The main feature of these means of transportation is that they authorize users to use vehicles *spontaneously* – without notifying the system operator in advance – to complete a one-way trip. Vehicles are taken at any station and returned at any station: when a user rents a vehicle at a station, he does not have to inform the operator about where and when he intends to return the vehicle. In exchange, he is not ensured to find a place to park his vehicle at his target station anytime.

The huge flexibility enabled by such means of transportation leads to imbalances in the vehicles repartition over the days. Some areas of the cities could suffer from a shortage of parking places, while in others users hardly find vehicles at stations. These imbalance issues

can occur in different places of the city depending on the time of the day.

Herbert Simon, 1975 Turing Prize and 1978 Nobel Prize in Economics, gives [81] a definition of a *complex system* in 1962 as a system "made up of a large number of parts which interact in a nonsimple way. In such systems, the whole is more than the sum of the parts, not in an ultimate, metaphysical sense, but in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole". Shared transport systems fit perfectly into the former definition: vehicles are moved by users, modifying the repartition of them in the network; interactions between users cannot be modeled in a tractable way: a user's decision to rent a vehicle at a specified station obviously impacts on the number of vehicles at this stations, but not only: if he takes the last vehicle, the station is empty. A future user arriving at this station would then have to roam through neighboring stations to find a vehicle. This shows that a user's decision does not impact only on his departure and arrival stations, but may affect events over the entire network: another decision would not have created this shortage of vehicles. The way a single user's decision echoes on other stations is a complex process. Thus, the need of a simulator appears clearly. It enables to model the shared transport system and to compare different strategies to cut the number of shortages or excess problems, plugging them into the simulator.

Regulation operations could help preventing such events to occur. Two types of operations are distinguished:

- The operator introduces a way to move vehicles from a station to another. In the case of a bike sharing system, bikes can be transported by trucks with a capacity on the number of bikes they can move at the same time. In the case of a car sharing system, employees could move a car from a station in excess to a station with a shortage of car. Such employees could be seen as "unit-capacity trucks". Instructions are given to these "trucks" on where to go and what to do. In Brussels Villo! system, 3 trucks are driven through the city during daytime, operating on the 180 stations.
- The operator does not plan to add any balancing operations which would need external factors. Incentive policies are considered to encourage users to regulate the system, preventing stations from being full or empty. In the Vélib' system, such system is already partially set up, since users who park their vehicles at specified stations receive free biking time. These stations are mainly situated on hills or at the periphery. For a car sharing

system, the solution of having drivers balancing the system seems to be very costly so other means to improve the system efficiency are relevant.

The first type of regulation strategies are easy to be implemented, while the second is harder. Operators attach importance to the *KISS – Keep It Simple & Stupid* – principle. It means that these means of transportation have to be easy to use. To a certain extend, having prices which change regularly could be seen as sticking out the former law.

The model of a shared transport system is described in Section 4.2. The simulator which was implemented is outlined in Section 4.3. Note that further instructions on how to download, install and use the simulator are available on the website later quoted.

4.2 Shared transport system: the model

In this section, the model used to represent a shared transport system is presented. In Subsection 4.2.1, the transportation offer is outlined. It refers to all the physical parameters of the system. The different features of the two types of regulation operations formerly mentioned are depicted. Subsection 4.2.2 describes how the users behave within the system.

4.2.1 Transportation equipment

A city with $n \in \mathbb{Z}_+$ stations is represented as a complete directed graph $G = (V, A)$ in which $V = \{1, \ldots, n\}$ is the vertex set, each vertex representing one of the n stations, and A is the set of arcs representing the shortest path in the city. The total number of vehicles is denoted by $N \in \mathbb{Z}_+$. Each vertex i has a capacity $C_i \in \mathbb{Z}_+$ in vehicles, which is the capacity of the station it represents. At each station the number of vehicles and parking places available over all the stations of the network are displayed. The time needed by a vehicle (resp. by walking) to go from vertex i to vertex j is a random variable T_{ij}^v (resp. T_{ij}^w), whose law depends on the context. The time needed to rent or to park a vehicle at a vertex is assumed to be negligible.

In the case regulation using trucks is tried, a fleet of $M \in \mathbb{Z}_+$ trucks is available to load or unload vehicles at the stations. Each of them has a capacity $K \in \mathbb{Z}_+$. These trucks are waiting for instructions from the operator. These instructions should include both the station where to go and the number of vehicles to load or unload there. Here again the time needed by a truck to drive from a vertex i to a vertex j is a random variable T_{ij}^t whose law depends on the context. The time needed to load or to unload vehicles at a vertex is assumed to be negligible.

In the case of a regulation without trucks, we assume that users have access to a reliable information system. The system displays *prices* which are updated regularly. These prices correspond to the incentive policies mentioned earlier. A user who parks his vehicle at a station will be charged this extra cost. The price a user will pay once he parks is the price he saw when he rents the vehicle: prices are frozen for a given time. Note here that users can also decide not to take the vehicle and to walk to their target vertex. Moreover, if prices are negative, the user will gain money – or free using time. The idea is to deter people from parking their rented vehicle at stations which are full or nearly full. Each user arriving at vertex i has a *target vertex* j. When he rents the vehicle, he is made aware of the prices over all the stations and could choose to park to another vertex rather than his target one. This choice is the result of an inner "thinking process" outlined in the following subsection.

4.2.2 Users behaviour

The users are assumed to arrive independently at a vertex i according to a Poisson process of parameter $\lambda_i \in \mathbb{R}_+$. The parameters of this law are assumed to be homogeneous over time. Each user arriving at vertex i has a target vertex j drawn at random with probability p_{ij}. These probabilities are kept in the Origin-Destination (O-D) matrix.

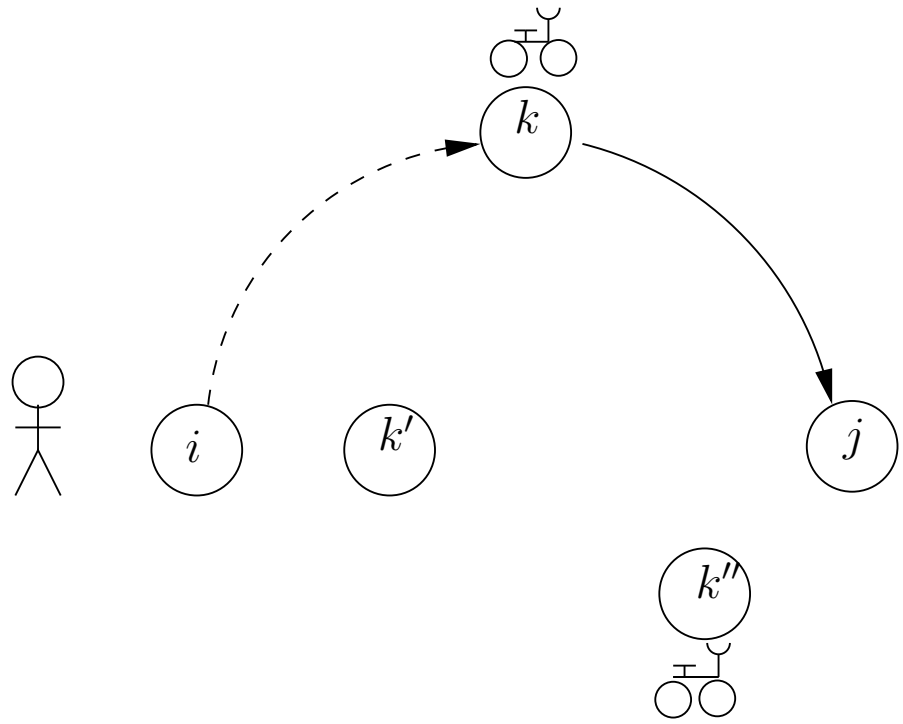

Figure 4.1: Choosing an alternative starting station

When a user arrives at an empty station to rent a vehicle, he is made aware of the state of each station in the system. He chooses to explore the station having vehicles that minimizes its total journey time, hoping to find a vehicle there, as it has been noted by [75] confirmed by practical studies. Figure 4.1 shows an example where a user shows up at station i willing to find a vehicle to go to station j. As station i is empty, he walks to station k where he knows there is an available vehicle. He hopes to find it once he arrives there to continue his trip to his destination. He is not going to station k' though it is closer to station i as he knows there is no vehicle available there. He is not going to station k'' because walking to k'' to rent a vehicle to finish the trip to j would result in a lengthier trip. The same behaviour occurs when looking for a parking place. Users choose to explore nearby stations with available parking places.

To avoid having users endlessly roaming through the city, two bounds are given for these two exploration phases. When he is looking for a vehicle, a user has a maximum number of vertices he is willing to explore and a limit on the time this exploration can last, after what he leaves the system unsatisfied. A similar two-bound limit is set for the exploration phase in the case he does not find a parking place. In the case a user leaves the system because he did not succeed in finding an available parking place within the limits of his exploration "patience", the vehicle is "lost". These two bounds are introduced with the intention of having a realistic system. Users who cannot find a vehicle after having explored a few stations opt for another transportation mean. If they cannot find a parking place, they leave the system with their vehicle, decreasing the total number of vehicles available within the system. This is indeed the case in Paris where after 4 years, none of the available vehicles are original ones [39]. These bounds on the exploration enables also to compute indicators – see Section 4.3.1.

When there is a pricing strategy, we have at time t a price $c_j(t) \in \mathbb{R}$ attached to each vertex j, which is the cost of parking a rented vehicle at the station. When a user starts to travel from a vertex i to a target vertex j at time t, he chooses an intermediate target vertex $k \in V$ which minimizes the next quantity that computes the *disutility* of his trip:

$$\texttt{dis}_k = \text{PRICEVHC} \times \mathbb{E}(T^v_{ik}) + \text{PRICEWALK} \times \mathbb{E}(T^w_{kj}) + c_k(t)$$

where $\mathbb{E}$ is the expected value of the transportation time. The coefficients PRICEVHC and PRICEWALK monetize the cost assumed by a user for using a vehicle or walking for a time period. They define the user's willingness to pay for saving time by using a vehicle. If the disutility is minimum for $k = j$, the user does not modify his target station. If the minimal disutility is achieved when the intermediate vertex k is the departure vertex i, the user walks to the target

vertex and rejects the system. However the number of rejections have to be carefully taken into account. Figure 4.2 illustrates the former thinking process. Solid line arrows correspond to trip

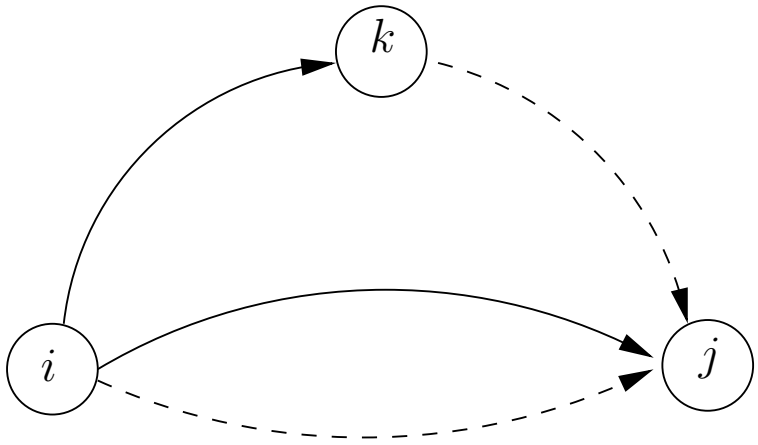

Figure 4.2: Choosing an alternative destination station

using a vehicle while dotted line arrows to walking. Assume a user shows up at station i where there is a vehicle available and wants to reach station j. Three alternatives are open for him. There are here listed and the price he would pay for it is given.

- The user rents a vehicle at i and directly goes to station j. The disutility is then $\texttt{dis}_j = c_j + \text{PRICEVHC} \times d_{ij}$.
- The user rents a vehicle at i and goes to an alternative arrival station $k \notin \{i, j\}$ from where he walks to station j. The disutility is then $\texttt{dis}_k = c_k + \text{PRICEVHC} \times d_{ik} + \text{PRICEWALK} \times d_{kj}$.
- The user decides not to rent the vehicle and walks to station j. The disutility is $\texttt{dis}_i = c_i + \text{PRICEWALK} \times d_{ij}$.

It seems that users who do not rent any vehicle should not pay any price c_i. However, the λ_i assume that all users actually rent a vehicle. This is the limit of the former method: it cannot be used to compare with other transportation means such as subway, taxi or walk. The calculated prices c_k only have meanings when comparing them together. The results are obtained give-or-take a translation on all the values. Adding EUR 1000 to all prices does not change the number of rejections. Therefore, OADLIBSim is unable to evaluate modal switching. Translating these penalties into "real prices" to compare the shared transport solution with other means of transportation would be a socio-economical topic.

All these parameters define different profiles of users. Several profiles can exist simultane-

ously. The proportion of each profile of users is specified. For instance, Table 4.1 describes four profiles of users. "Nervous" users explore only 2 stations and spend 2 minutes maximum to find both a vehicle or a parking place, while "Patient" users can visit up to 8 stations (respectively 10 stations) and spend up to 20 minutes (resp. 25 minutes) to find a vehicle (resp. a parking place).

Type of user	Proportion (in %)	Max # st. for a vehicle	Time lim. for a vehicle (in min)	Max # st. for a parking	Time lim. for a parking (in min)
Nervous	30	2	2	2	2
Impatient	20	2	2	4	8
Reasonable	10	4	10	7	15
Patient	40	8	20	10	25

Table 4.1: Example of different user profiles in the simulator

All the former features need to be taken into account in the simulator. Its objective is to offer simple methods to add both type of strategies and to test them over several instances. The issue of how to evaluate the performance of a system has also to be taken into account.

4.3 Description of the simulator

In this section, the simulator is presented and the question of finding appropriate indicators is addressed. The simulator takes as input all the parameters described before and enables to compare the performances of different management strategies. It copes with huge dimensions. Several relevant indicators to evaluate the performances of the system.

The simulator OADLIBSim utilizes OMNeT++, a discrete event simulation environment. It enables to successfully simulate complex systems. Its flexibility made it a useful tool in other areas such as queuing network. The different components are programmed in C++. OADLIBSim runs on all platforms. It presents a user friendly interface. OADLIBSim is freely available on the website of the project : http://cermics.enpc.fr/∼meuniefr/home.html/OADLIBSim_Site

To simulate a shared transport system, one needs to specify the system characteristics to OADLIBSim. These characteristics are formed by a set of values, which gather the system parameters. There are simple values such as the number of stations or vehicles, and matrices such as the O-D matrix or the mean travel time to go from a station to another.

Additionally, one must enter the set of regulation methods to be tested. Default behaviour for OADLIBSim is to launch regulation-free simulations. By using the simulation inputs, regulation methods can be enforced using trucks, prices or both. These methods can be easily added into the simulator. They have to be programmed in C++. A template for each type of methods guides the implementation of a new component.

The simulation inputs are defined using a single parameter file (usually named with the file extension .ini), and a set of .csv files. The .ini file contains all simulations single-valued parameters such as for example size, vehicles and parking number... It indicates the .csv files to be used for matrices parameters. Table 4.2 lists all the input information with respect to the order they appear in the .ini file. The three last inputs are not mandatory. If not precised, default parameters are taken to compute the truck travel times and to define a user's profile. Stations coordinates are used only to show a coherent display in the graphical interface. Then, a boolean value indicates whether log files are needed or not. They are mandatory in the case we want to draw figures such as those shown later in this section.

Another software can be used to generate inputs. CityBuilderGui is its graphical front-end. It allows the specification of all the needed data to generate a city. It enables to view the produced network so to check the quality of the stations distribution.

Different regulation strategies can be easily added and programmed in C++. Each strategy is a subclass of a virtual class, either the one using trucks or the one with prices. When programming a new strategy, one has to give a name to this subclass and to implement the few methods which have to be written thanks to the template formerly mentioned. In the case of a truck regulation strategy, *missions* are given to the trucks. A mission is a sequence of stations to visit with logistic operations to complete. Two types of missions can be given to trucks: *default mission* instructions are "Drive to station s and load t vehicles there". The truck drives to the station where it would try to load the ordered number of vehicles, whatever the number of vehicles parked at the station. If this number is less or equal to t, the trucks empties the station; *objective driven mission* instructions are "Drive to station s and move the number of parked vehicles as close to an objective o as possible". The truck travels to the station where it tries to equal the number of parked vehicles with o, leading to loading or unloading operations.

To add regulation methods, further information has to be specified in the .ini file. They are listed in Table 4.3. Note that pricing and truck regulation methods can be used at the same time.

Once all parameters loaded by OADLIBSim, one has to specify which regulation strategy to use. The strategy to be tested has to be chosen within the list showing up as in Figure 4.3. The

Name of the inputs	**Description of the input**
Simulation time limit	the limit on the simulation time can be expressed in hours, minutes and seconds
Number of repeat	the simulation can be repeated several times with a different seed in order to test the robustness of the strategies
Seeds for the different random components	OMNeT++ simulation enables to have different seeds for different components simulating discrete events that are not linked
Number of stations	number of stations present in the system
Number of vehicles	the number of vehicles initially present in the system; when the simulation begins, they are all parked at stations and their initial repartition is uniform over all the stations
Number of parking	the total number of parking places
Number of trucks	the total number of trucks
Capacity of a truck	maximum number of vehicles on a truck
Vehicle travel time file	name of the .csv file keeping the mean vehicle travel time matrix; the mean time to walk from a vertex to another is taken as the mean time spent by a vehicle for the same displacement multiplied by a constant factor
OD file	name of the .csv file keeping the O-D matrix
Arrival rate file	name of the .csv file keeping the station client arrival rates. They are homogeneous
Station capacity file	name of the .csv file keeping the capacity of each station
Station coordinates file	name of the .csv file giving the three coordinates (x, y, z) of each station
Truck travel time file	name of the .csv file keeping the mean truck travel time matrix
Client categories file	name of the .csv file keeping the different user profiles and their proportions

Table 4.2: Inputs of OADLIBSim

Name of the inputs	**Description of the input**
TrucksManagerVersion	name of the class used for the regulation method
TrucksRequestMission	boolean indicates whether the truck request mission or not; once a mission is finished, the truck can either request a new mission or wait for the operator to send him a new mission
TrucksMissionObjectiveDriven	boolean indicates whether the truck mission type is default or objective driven
UnrequestedMissionInterval	time after what a new mission is given to trucks; in the case this optional input is activated, new missions are given to trucks at each time slot; in the case a truck already has a mission, its old mission is erased and the truck follows the updated instructions
PricingEnforcement	boolean indicates a pricing strategy exists
PricingUpdatePeriod	time between two successive updates on the prices
PricingManagerVersion	name of the C++ class of the strategy

Table 4.3: Further inputs for regulation method

default strategy named “TruckMissionBuilderDummy” does not give any instruction. Trucks do not interfere with the system. Vehicles are neither loaded nor unloaded.

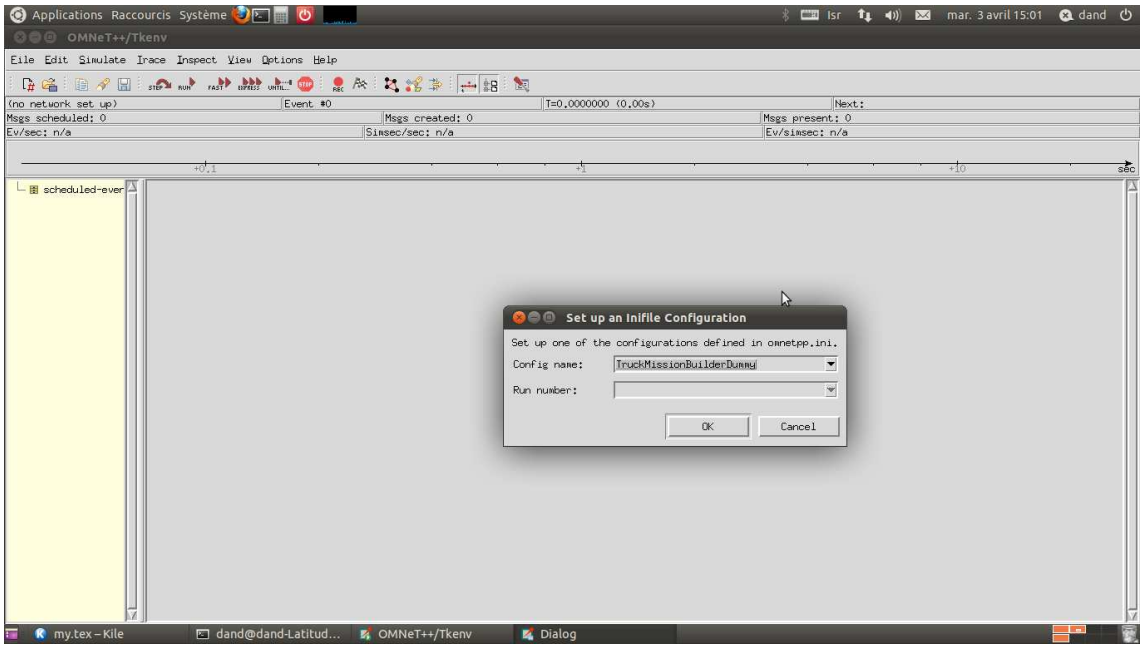

Figure 4.3: Interface: choosing the strategy

Then, the system is loaded and by clicking on the Play button, the simulation starts. “Events” are created. An event is the arrival of a user at a station, looking for a vehicle or a parking place. A user that is able to find a vehicle is then sent in the “Itinerary Manager” during the time of his trip. This is shown in Figure 4.4 where a bike logo leaves a station for the logo representing the “Itinerary Manager”, in the bottom right. In this case the network has only five stations and the user takes a vehicle at station 2. When the user arrives at his final destination station, the same bike leaves the “Itinerary Manager” to go to the corresponding station. Figure 4.5 illustrates this case with a network of 125 stations. If he does not manage to park there and that the user roams, the bike returns to the “Itinerary Manager”.

When the regulation strategy uses trucks, they also follow the same trail: when leaving a station for another one, a truck is sent to the “Itinerary Manager” during the time he needs to drive to his destination. Figure 4.6 shows an example with the five station network.

Note that the time elapsed for the regulation strategy to complete is inserted in the simulation. In the case of regulation using trucks, when a truck has no instruction, the method is run to provide him new instructions. Before calling the method, the simulation is stopped. Then, if the computing time is a minute, the simulation is run for a minute during which *the truck does*

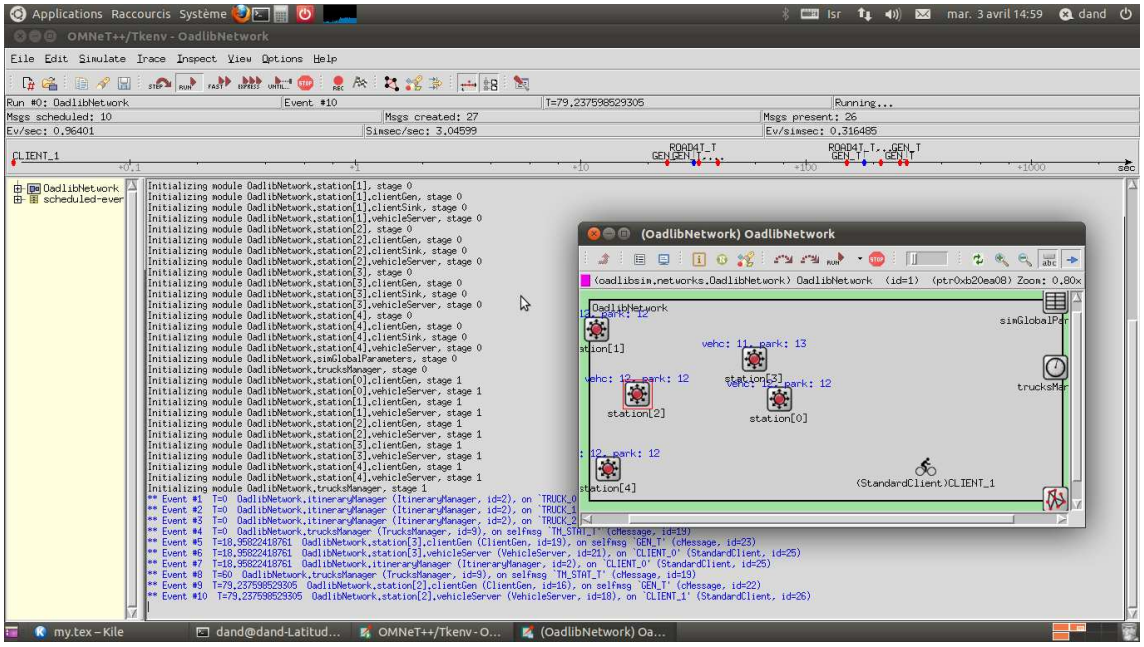

Figure 4.4: Interface: simulating a trip leaving station 2

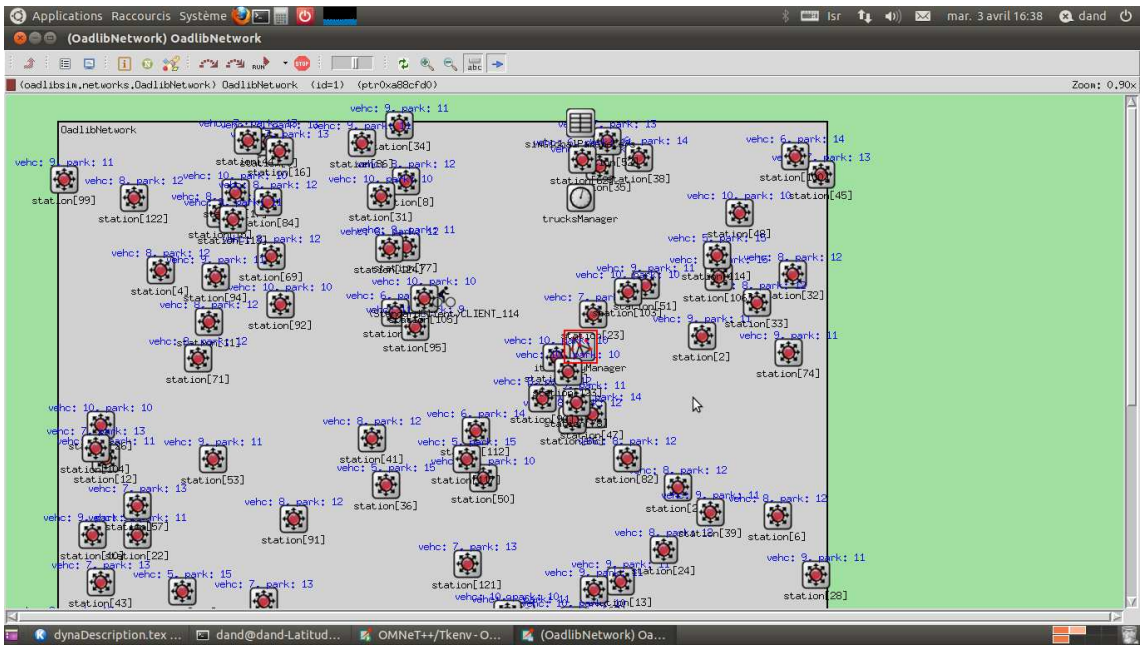

Figure 4.5: Interface: arriving at a station

not move. After a minute in the simulation time, the truck receives his new instructions and starts to complete them. This ensures to take into account the time needed by the regulation methods.

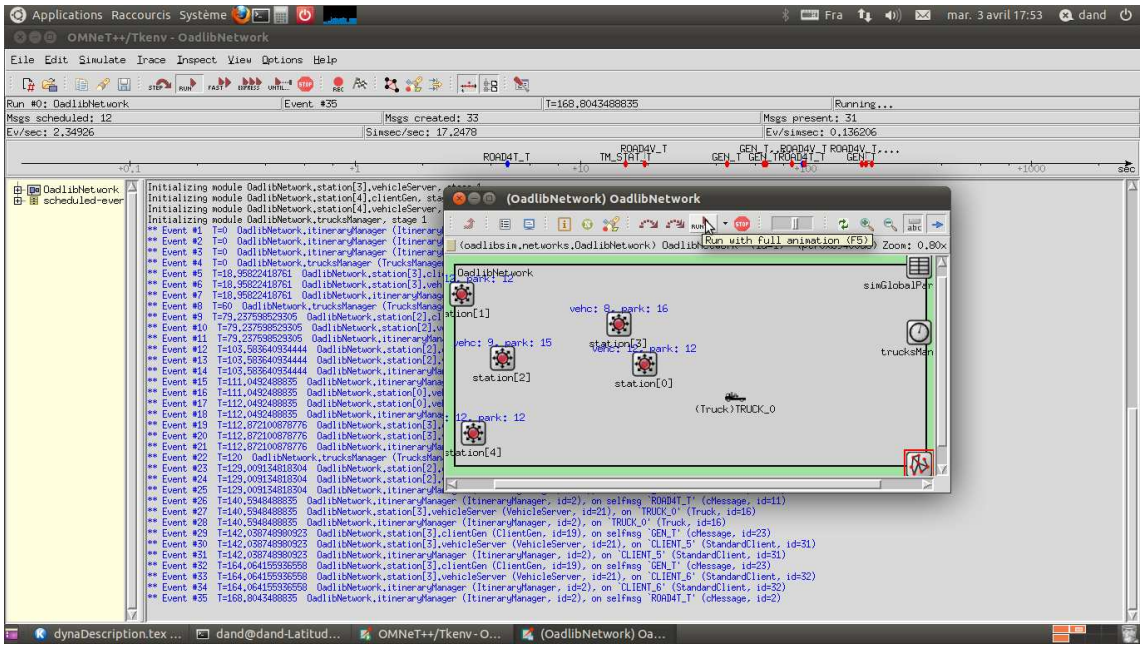

Figure 4.6: Interface: regulation using trucks

4.3.1 Evaluation of the quality of the management

Our purpose is to compare different strategy to enhance the system efficiency by improving regulation strategies. In the present subsection, we want to design various indicators allowing to evaluate and compare management strategies and algorithms. We emphasize that in our model, the demand is inelastic: the parameters of the Poisson laws λ_i and the O-D matrix p_{ij} are independent of the quality of the management strategies. It is of course an approximation, which is classical in such context, especially in transportation science. Global modelling is more or less out of reach. We can think for instance about the four-step model in transportation: the last one – traffic assignment – which has received a lot of attention – works often with a predetermined demand.

During the simulation, information is kept and they can be displayed at the end of the simulation. For instance, Figure 4.7 displays the evolution of the vehicles when there is no regulation. The network has 125 stations each of which has 20 parking places. Initially 1250

vehicles are available. After a short transition period, half of the vehicles seems to be in use anytime. The third curve which is slowly increasing is the number of "lost vehicles". In that example, about 100 vehicles are lost. In that simulation, users are all chosen to have 0 patience: whenever they do not find an available parking place at their target station, they leave the system with their vehicle. Information can also be displayed per station. Figure 4.8 shows the evolution of available parking and vehicles at the station 6 in the former network case. These two curves are symmetric. The third curve represents the number of vehicles lost at the station. This curve increases at the end of the simulation period. With no surprise, it increases when the station is full. Users arriving at the station at this time cannot park their vehicles and so they leaves the system with it. At the end of the simulation, 7 vehicles are lost at this station.

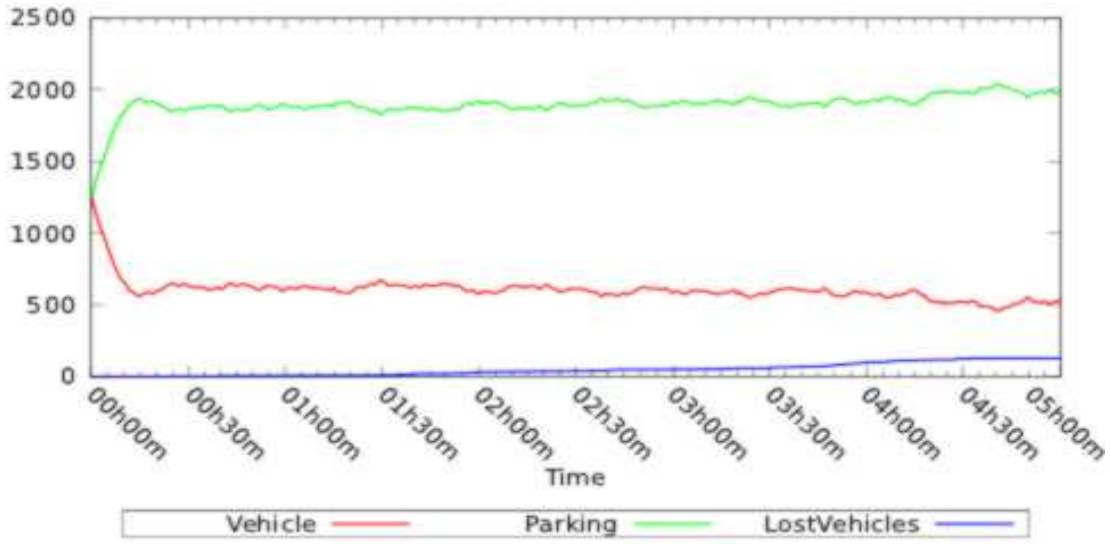

Figure 4.7: Example of the type of output for the network

Indicators have to be defined to compare the performances of the different strategies. They should answer the two following question: how many users have found a vehicle ? How many users have found a place to leave their vehicle ? As the objective is to measure the system efficiency with respect to the service is to count the number of *satisfied* users. A user is said to be satisfied when he manages to rent a vehicle at a station and park it at a station. If he roams to find a vehicle or a parking within his patience limits, he is still considered as satisfied. The log files produced at the end of the simulation enable to discriminate between the number of exploration steps users have to do before finding a vehicle or a parking.

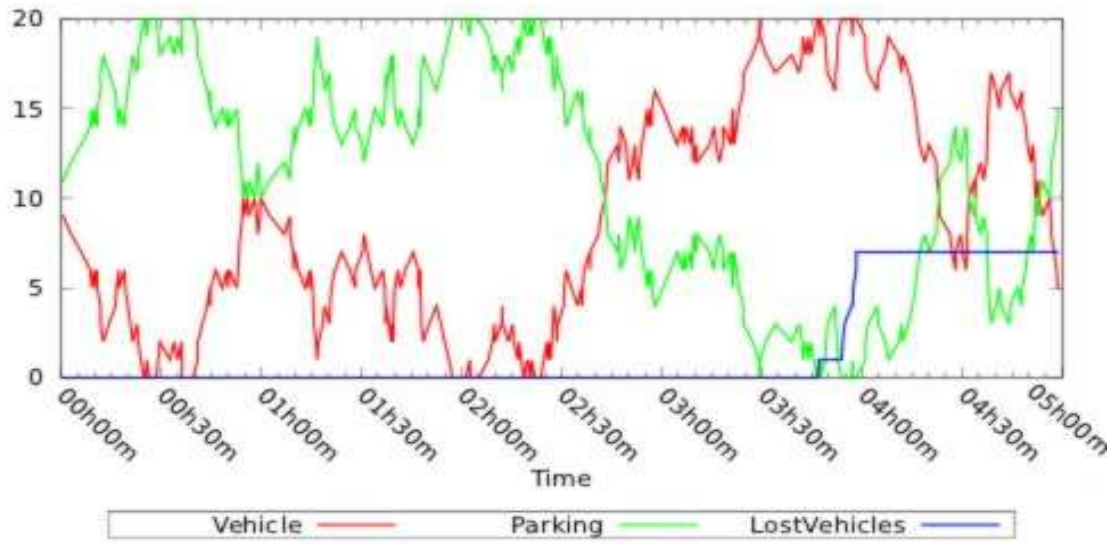

Figure 4.8: Example of the type of output for a station

If there is a regulation strategy using trucks, the efficiency of the strategy could be measured comparing the number of satisfied users obtained with this regulation with the number of them without regulation. The gain of satisfied users divided by the total number of vehicles moved by the trucks during the simulation could give a measurement of the average impact of moving a vehicle. One can think that this ratio is bounded by 2, as moving a vehicle from a station to another would enable one user to park its rented vehicle at the departure station, and another user to rent the unloaded vehicle. However this idea is not completely correct. Figure 4.9 shows part of a network, with $k+2$ stations. The values of the O-D matrix p_{ij} are displayed next to each arc. For any $t \in \{1, \ldots, k-1\}$, we assume that $p_{s_t s_{t+1}} = 1$. Assume that vehicles are initially parked at station p. The thick arrow stands for a trip made by the truck moving vehicles back from station d to station p. Each vehicle brought back would cause in average a gain of $0.7 + 0.3k$ satisfied users. When k increases, this average gain exceeds 2.

If there is a pricing strategy, an indicator is the maximal magnitude of the prices while regulating the system. In that case, the users that rather walk from their initial station to their destination are counted separately.

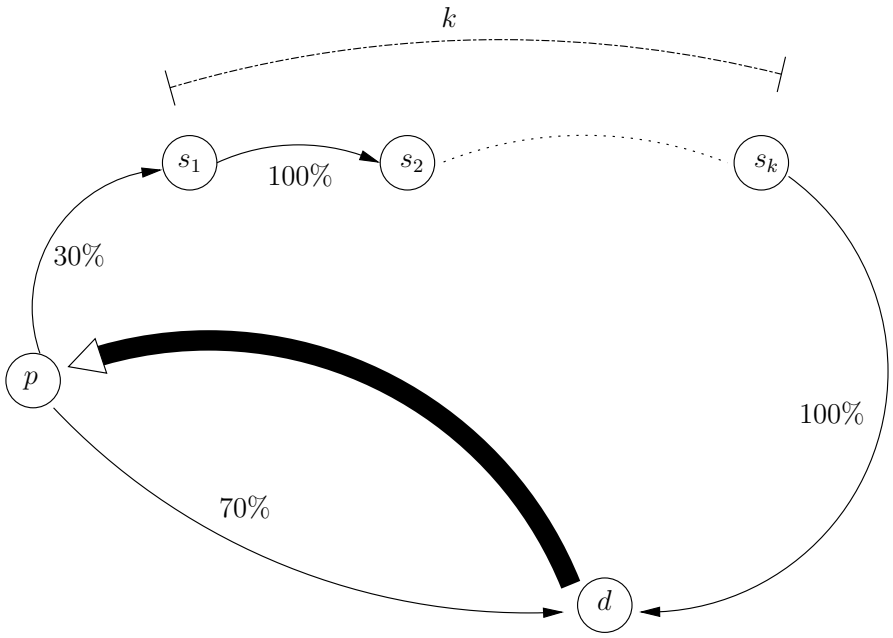

Figure 4.9: Indicator: about the efficiency ratio

4.4 Conclusion

In this chapter, we presented the model of shared transports used and also the simulator OADLIBSim. This simulator can be downloaded at the website of the project. It enables people to easily tune the simulation environment by specifying the network scheme and parameters and the users behaviour in input. New management strategies can quickly be added in the simulator and tested. The different indicators which can be computed to estimate the performances of these strategies are given.

Chapter 5

Real time optimization methods

5.1 Introduction

In the Parisian Vélib' system, several methods have been used since the system started in 2007. 23 trucks and 3 buses are operating in the city days and nights. Incentive policies with the V+ stations were also introduced to encourage users to return their rented vehicles at some stations regularly suffering from a shortage of bikes. Users obtained a compensation – free riding time – when bringing a bike from a "normal" station to one of these stations, that are mostly situated on hills or at the periphery.

Despite all the means that were introduced, imbalance problems keep occurring in network. In "Le Figaro" article [12] in 2008, Albert Asseraf, Strategy and Marketing France Chief Executive at JCDecaux in Paris confesses that *it is statistically impossible to be sure to find a bike or a parking place in 100% of the cases*. It is difficult to figure out a way to ensure to all users that they will be able to find a bike in their departure station or a rack in their target station. A good regulation method should have bikes or racks reachable in reasonable time, if not available everywhere. To that purpose, the instructions which are given to the drivers of the trucks have to be cleverly chosen. A system in which all truck drivers would take their own decisions would highly depend on their experiences and may end up with more imbalance problems. Moreover, if their actions are not coordinated and they take their decision looking at a map showing the current situation in the network, they could play one against the others: for example, if two drivers see a station nearly empty, they both drive there to unload vehicles; or if a driver sees a station nearly empty and drives there and unload vehicles, another driver could then see the same station now supplied with vehicles as nearly full and drives there to

load the vehicles the first driver has just unloaded; or if a driver chooses to drive to an empty station in the network, but by the time it arrives the station is full. These examples show the need for a unique and central monitor center.

The objective is to find algorithms for providing instructions to drivers to improve the level of service. Two types of methods are experimented: the first one gathers algorithms using trucks, the second one uses a pricing strategy. In this chapter, different heuristic methods are presented. All of them are *objective driven missions* (see Section 4.3 of the former chapter): for each station, a target occupation level is determined by the operator. The problem of computing this value is not addressed here. In Section 5.2, all the methods use trucks. Section 5.3 proposes a method using only incentive policy and no truck. Section 5.4 enables to compare the different results obtained running these different methods on the same instances. Comments are added to explain the performances of each of the method. At last, Section 5.5 proposes a conclusion on this chapter.

Before starting the description of the different methods, few definitions that are used throughout this chapter are given. Finding the optimal action to give to the trucks drivers should take into account all the system, leading to a huge number of variables to take into account as it is discussed in Subsection 5.2.4. This is not feasible in real time where instructions should be quickly given. Two alternatives can then be followed: either the number of variables are reduced, or we fix short-term objectives. For that purpose, we introduce for each vertex $i \in V$ its target occupation level $\theta_i \leq C_i$. In all the method presented here, the objective is to have the level of occupation of the stations staying around these values. Recall that λ_i is the parameter of the Poisson law representing the user arrival, and that p_{ij} is the probability for a user who arrives at station i to have station j for destination. The p_{ij} are kept in the O-D matrix. The number of vehicles parked at a station i is denoted as x_i, with $0 \leq x_i \leq C_i$. If forecasts are done on the future number of vehicles, it is denoted with $\widetilde{x}_i$ with $0 \leq \widetilde{x}_i \leq C_i$.

5.2 Methods using trucks

5.2.1 Preliminaries

In 2010, George and Xia [48] proposed the following model. The system is seen as a single-class closed queuing network whose nodes are of two types: the station-nodes, representing the

stations, and the trip-nodes, representing the trips between the stations. Each station-node i is connected to all trip-nodes (i, j) and each trip-nodes (i, j) is connected to station j. In this queuing network, the *customers* are the vehicles, which have to be distinguished from the users. A station-node i is a $M/M/1/C_i$ queue with service rate equal to λ_i: vehicles are served by the users. A trip-node (i, j) is a $M/G/\infty$ queue with service time T_{ij}^v.

Note that in this model, if a vehicle arrives at a station i with no empty rack, it disappears from the system. A way to circumvent this drawback consists in deleting the capacity C_i in each station i, but this time, each station gets an infinite capacity. An exact model, taking into account the rerouting of the users when they do not find vehicles or parking places in a purely mathematical model is anyway out of reach.

Regulation using trucks is the easiest one to introduce in cities. This explains the fact that it is widely spread over all the BSSs. However, finding good regulation methods is a ceaseless worry of the operators. In some cities, these trucks operate 24 hours per day. In this section, we propose now some simple heuristics. More sophisticated algorithms are not tested since between two tasks of a truck, the system is assumed to change quite a lot. All of them give missions to truck drivers. Once they are done with their mission, they ask the central monitor for a new one. They all work with objective driven missions (see Chapter 4.3).

Before describing these heuristic methods, we give a few results to show the complexity of the problem.

Proposition 5.2.1. *If the arrivals follow exponential laws and there is only one truck, the problem of finding the action of the truck maximizing the probability for the first user to find a vehicle is $\mathcal{NP}$-hard.*

Proof. Let $G = (V, E)$ be a graph, $|V| = n \in \mathbb{Z}_+$. Define $G' = (V', E')$ another graph such that $V' = V \cup \{o\}$ the vertices and $E' = E \cup \{(o, v), v \in V\}$ the edges. G' is G to which a vertex o is added and linked to all the vertices of G. Assume that a truck is initially parked at o with n vehicles loaded. V is the set of stations, which are all initially empty. Users arrive at each station with respect to a Poisson process with a parameter $\lambda \in \mathbb{R}_+$. It takes exactly one unit of time to traverse any edge of the graph G'. Assume that we are able to maximize the probability to catch the first user, where catching a user means that there is a vehicle available at the station where he shows up.

Dealing with n independent Poisson processes with parameter λ is equivalent to deal with a unique Poisson process of parameter $n\lambda$ giving the arrival of a client in the system, followed by

the selection of a station uniformly at random. We will here prove that if there is an Hamiltonian chain in the graph, the strategy maximizing the probability to catch the first user consists in following this Hamiltonian path. Any other choice would lead to a lower probability to catch the first user.

Suppose there is an Hamiltonian chain in the graph G, then, following this chain, the truck can unload 1 vehicle at each time step. The probability that the first arrival time t_a is lower or equal to any $t \in \mathbb{R}_+$ is $\mathbb{P}[t_a \leq t] = 1 - e^{-n\lambda t}$. If the first user arrives before 1 ($t_a < 1$) he is not served for sure as the truck would not have had enough time to reach any station. If $1 \leq t_a < 2$, the truck would have only had time to visit one station. So in $n-1$ out of n cases, he is not served. Repeating the argument, we obtain that the probability of missing the first user while following the Hamiltonian chain is:

$$P(n,\lambda) = \sum_{k=0}^{n-1} \frac{n-k}{n} \mathbb{P}[k \leq t_a < k+1] \tag{5.1}$$

$$= \sum_{k=0}^{n-1} \frac{n-k}{n} \left(e^{-n\lambda k} - e^{-n\lambda(k+1)}\right) \tag{5.2}$$

$$= \sum_{k\geq 0}^{n-1} \alpha_k \beta_k \tag{5.3}$$

where $\alpha_k = \frac{n-k}{n}$ is the ratio of unvisited station at time step k over all stations and $\beta_k = e^{-n\lambda k} - e^{-n\lambda(k+1)}$ is the probability for the first arrival to occur between k and $k+1$.

If there is no Hamiltonian chain, then we cannot visit a new station at each time step. With the same lines of reasoning, we get a probability $P'(n,\lambda) = \sum_{k=0}^{\infty} \alpha'_k \beta_k$ with $\alpha'_k \geq \alpha_k$ for all k and $\alpha'_k > \alpha_k$ for at least one k_0. Thus $P'(n,\lambda) > P(n,\lambda)$.

We have then proven that there is an Hamiltonian chain in G if and only if the minimum value of the probability of missing the first user is equal to the value $P(n,\lambda)$. As the problem of determining whether it exists an Hamiltonian chain in a graph is $\mathcal{NP}$-complete, the one of finding the strategy maximizing the probability to catch the first user is $\mathcal{NP}$-hard. □

Proposition 5.2.1 proves the difficulty of ensuring to an operator that the proposed solution is the best one. If now the objective is not to find the first user, but several users, the problem is not a repetition of the former problem. The case the objective is to find the strategy which maximizes the probability to catch the two first users is outlined below. Here also catching a user means that the station where he arrives has an available vehicle. First, we explain why this problem is not the same as finding the strategy which maximizes the probability to catch

the first user used twice. To that purpose, assume that there is a network of n stations of unit capacity with $n-1$ stations gathered together – the "central" stations – and a last station far from the others. Assume that the truck is initially at a depot near to the central stations, and that it takes exactly one time unit to the truck to traverse any edge but those having the isolated station for endpoint, in which case the travel time is substantially longer. Lastly, assume that the truck has visited the $n-1$ central stations, where it has unloaded one vehicle, and that the first user is still not arrived. If the objective is to maximize the probability to catch the first user only, the truck is sent to the last isolated station. If the objective is to maximize to probability to catch the two first users, it could be better to have the truck staying around the group of central stations waiting for the first user's arrival. If the first user arrives at one of these station and rents a vehicle, the truck travels there to unload a vehicle before going to the last isolated station. If he arrives at the empty isolated station, he cannot be served. The probability that he would arrives at the isolated station is low ($\frac{1}{n}$). Depending on the mean arrival rate λ with respect to the travel time, this strategy can be better.

To deal with the two first user case, several definitions need to be given. Let $G = (V, E)$ be a graph and $|V| = n$. G is said to have an *Updatable Hamiltonian Chain* (UHC) if G has an Hamiltonian chain $c = v_1, v_2, \ldots, v_n$ such that, for any $1 \leq j < i \leq n$, the graph $(V_{i,j}, E[V_{i,j}])$ where $V_{i,j} = \{v_j\} \cup \{(v_t), t \geq i\}$ has an Hamiltonian chain. In a graph, a vertex is said to be *complete* if there is an edge linking it to all the vertices of the graph. We have now the two following lemmas:

Lemma 5.2.2. *A chain $c = v_1, v_2, \ldots, v_n$ is an UHC if and only if c is an Hamiltonian chain and v_n is a complete vertex.*

Proof. If $c = v_1, v_2, \ldots, v_n$ is an Hamiltonian chain, and v_n is a complete vertex, then for any $1 \leq j < i \leq n$, the chain $v_i, v_{i+1}, \ldots, v_n, v_j$ is an Hamiltonian chain in $V_{i,j}$. Then it is an UHC. Conversly, taking $i = n$ and any $j \leq n-1$, v_n has to be linked with all vertices of the graph, so it is a complete vertex. □

Lemma 5.2.3. *Determining whether an Hamiltonian chain exists in a graph without any complete vertex is $\mathcal{NP}$-complete.*

Proof. Assume there is a polynomial algorithm $\mathcal{A}$ that determines if there is an Hamiltonian chain in a graph without complete vertices. We are going to prove that for any graph $G = (V, E)$ the same algorithm $\mathcal{A}$ decides whether there is an Hamiltonian chain or not. Define $V = V' \cup V^C$, where V^C is the subset of complete vertices of V in G, and $V' = \{v_1, v_2, \ldots, v_k\}$

the other vertices. If the graph G is a clique, then V' is empty and so there is an Hamiltonian chain. Otherwise, for $i \leq k$, let $G_i = (V_i, E_i)$ be a graph such that $V_i = V \cup \{o\}$ and $E_i = E \cup \{(o, v_i)\}$. G_i has no complete vertex. So algorithm $\mathcal{A}$ can be run. Two possibilities can be faced:

- if there is an $i \leq k$ such that $\mathcal{A}(G_i)$ returns "true", there is an Hamiltonian chain on G_i. By construction vertex o has to be an endpoint of the chain. If it is erased, the same sequence of vertices would be an Hamiltonian chain on G, so it means that there is an Hamiltonian chain in G;
- if, for all $i \leq k$, $\mathcal{A}(G_i)$ returns "false", it means that there is no Hamiltonian chain in G. Indeed assume there is one. If one endpoint is not complete, then the $\mathcal{A}(G_{i_0})$ should have returned "true" where v_{i_0} is the endpoint. If both endpoints of the Hamiltonian chain are complete vertices, then they are linked by an edge and there is an Hamiltonian chain following the same edges starting from any vertex v_{i_0} that is not complete. In that case, $\mathcal{A}(G_{i_0})$ should have given true.

We have proven that if such an algorithm exists, it can also determine if there is an Hamiltonian chain in the general case. This proves that determining whether an Hamiltonian chain exists in a graph without complete vertices is an $\mathcal{NP}$-complete problem. □

Lemma 5.2.4. *Determining whether an UHC exists in a graph is $\mathcal{NP}$-complete.*

Proof. Assume that such a polynomial algorithm exists. It is able to determine whether there is an Hamiltonian chain in a graph $G = (V, E)$ without any complete vertex. Let $G' = (V', E')$ be a graph such that $V' = V \cup \{o\}$ and $E' = E \cup_{v \in V} \{(o, v)\}$. Vertex o is a complete vertex, it is is the only one in G'. Thanks to Lemma 5.2.2 it is proven that any UHC finishes at a complete vertex. So if there is a UHC in G', it ends at o. The rest of the UHC is then an Hamiltonian path in G a graph without any complete vertex. So finding an UHC in G' is equivalent of finding an Hamiltonian chain in G that has no complete vertex. The latter problem is proven to be $\mathcal{NP}$-complete in Lemma 5.2.3. So finding an UHC in a graph is also an $\mathcal{NP}$-complete problem. □

Proposition 5.2.5. *If the arrival follow exponential laws and there is only one truck and all the station capacities are unit, the problem of finding the action of the truck maximizing the probability for the two first users to find a vehicle is $\mathcal{NP}$-hard.*

Proof. Taking the same graph G as in the proof of Proposition 5.2.1, the best strategy appears to be a UHC. Indeed, the best the truck can do is to put one vehicle at each time step at a

new station. If the first user arrives at a station that was visited by the truck, but before it has completed its tour, the UHC ensures that there is an Hamiltonian chain on the remaining stations and the now empty station. If the first user arrives after the truck has visited all the stations, it waits at the last vertex, which is a complete vertex according to Lemma 5.2.2. Once the first user has shown up, the truck drives to the now empty station in one time step to be sure to catch the second user.

If the truck follows an Hamiltonian chain but not an UHC and the first user shows up at a station visited by the truck before he finished its tour, it is not sure that the truck could visit all the remaining stations and the now empty station following an Hamiltonian chain, and so a station would stay empty longer than in the UHC case. The probability of losing the second user would then be higher. So any other strategy would end with a greater probability to lose the two first users.

If we had a way to find the strategy maximizing the probability of catching the two first users, we then would know if the corresponding graph has an UHC. So this problem is $\mathcal{NP}$-hard according to Lemma 5.2.4. □

We have then proven that finding the action to execute to catch the first or the two first users are $\mathcal{NP}$-hard problems. We conjecture that it is still $\mathcal{NP}$-hard for more than the two first users. In the remaining of this section, heuristic methods are outlined.

5.2.2 One-step - two-step heuristics

A truck is at a station with a certain load. When it asks for a mission, the algorithm looks for the two most unbalanced stations (i.e. the ones with the greatest excess and the one with the greatest deficit with respect to their objectives θ_i). The truck is then sent to these two stations, depending on its current load. If the load on the truck is lower than a given threshold, the truck is sent to the station where vehicles are to be loaded and then to the station with a deficit. Otherwise, it visits the two stations in the reverse order. In the one-step heuristic, only the first move is sent to the trucks, which will ask for a new mission once the move realized.

5.2.3 One-step - two-step heuristics with forecast

The former method does not take into account any forecasts on the number of vehicles that are expected to be present at the stations once reached by the truck. It only uses the current

number of vehicles parked at a station. Simple techniques could give an hint on the number of vehicle arriving in our case computing the average return rate μ_j at a station j as follows:

$$\mu_j = \sum_{i \in V} \lambda_i p_{ij} \tag{5.4}$$

In the stationary state, the continuous arrival of users at stations to rent vehicles leads to a continuous flow of users returning vehicles. These μ_j give us the number of vehicles returned per minute at each station. Therefore, the number of vehicles expected at a station j, denoted $\widetilde{x}_j$, once a truck leaving i would reach it is computed as follows:

$$\widetilde{x}_j = Proj_{[0,C_j]}\left(x_j + \frac{1}{10}\mathbb{E}(T_{ij}^t) \times (\mu_j - \lambda_j)\right) \tag{5.5}$$

where T_{ij}^t is the time needed by a truck to drive from vertex i to vertex j and $Proj_{[a,b]}$ is the projection operator from $\mathbb{R}$ on the interval $[a, b]$: if $x < a$ (respectively $x > b$), then $Proj_{[a,b]}(x) = a$ (resp. $Proj_{[a,b]}(x) = b$). Otherwise, if $x \in [a, b]$, $Proj_{[a,b]}(x) = x$. The decisions are then made with the corrected number of vehicles at stations.

The $\frac{1}{10}$ factor seems to perturb the evaluation of the future number of vehicles present at the station. However, it seems that in practice the formula without this factor tends to over-evaluate the number of vehicles to be returned or rent at a station. This factor was tuned with respect to several experiments.

Three methods using the mean return rate per station μ_i: the *one-step heuristic with forecast* method, the *two-step heuristic with forecast* method and the *two-step one-stop heuristic with forecast* method.

One-step heuristic with forecast

For the *one-step heuristic with forecast*, the former formula is appropriate as it takes the expected time to reach the station. It is used to evaluate the imbalances at any stations. The two most unbalanced stations are kept and depending on the load of the truck, it is sent to the station expected to suffer from the most excess or deficit in vehicles.

Two-step heuristic with forecast and two-step one-stop heuristic with forecast

In this case, the expected number of vehicles at the two stations depends on the order they are visited. The truck is at a station $i \in V$. It could first go to station $k \in V$ among the $n - 1$ stations for the first stop, and from there to station $j \in V$ among the $n - 2$ remaining stations for the second stop (it is not allowed to come back to its origin station i).

At each stop, the objective is to bring the number of vehicles closer to the station target θ_i. As the initial load on the truck l is known, the action to be achieved at each stop can be anticipated. If for instance $\widetilde{x}_k$ is larger (respectively smaller) than θ_k, the truck load after the first stop would be $l' = l + \alpha_k$ (resp. $l' = l - \alpha_k$), where $\alpha_k = \min\{K - l, \widetilde{x}_k - \theta_k\}$ (resp. $\max\{l, \theta_k - \widetilde{x}_k\}$). At the second stop, the number of expected vehicles at station j is evaluated as follow:

$$\widetilde{x}_j = Proj_{[0,C_j]}\left(x_j + \frac{1}{10}\mathbb{E}(T^t_{ik} + T^t_{kj}) \times (\mu_j - \lambda_j)\right)$$

where the time takes into account the first stop at station k. The expected action to be achieved there can then be evaluated with the same reasoning. By summing α_k and α_j, we obtain the total number of vehicles expected to be moved with respect to a sequence of station. The instruction sent to the truck is to follows the sequence of stations maximizing $\alpha_k + \alpha_j$.

In the *two-step heuristic with forecast* method, the truck is given the sequence of stations and complete both visits. In the *two-step one-stop heuristic with forecast* method, the truck is only sent to the first stop. The second stop is used to make the decision, but the length of the truck mission is only one. It is a way to take into account the future state of the network in the decision.

5.2.4 The Colored Cluster heuristic

This method aims at finding the optimal instruction to give to truck drivers at a certain point knowing the system. Even if the results are not satisfying, we put it here since it contains ideas that may help in others methods as it takes into account several important features of the problem. We first give some hints on the method applied here, the optimal policy iteration algorithm which can be used to solve the Stochastic Shortest Path Problem (SSPP). The SSPP is one of the subjects treated in Tsitsiklis and Bertsekas book [17]. This problem and its resolution method are presented in the next subsection. To use this method, several approximation are outlined, so it is a heuristic method.

Stochastic Shortest Path Problem

We introduce here some notations. If we have a discrete-time Markov chain, $\mathcal{E}$ represents the states of the system. At each state $i \in \mathcal{E}$, a control has to be chosen within a given finite set $U(i)$. The choice of a control $u \in U(i)$ at a state $i \in \mathcal{E}$ specifies the transition probability

$l_{ij}(u)$ to the next state j. For each couple of state $i, j \in \mathcal{E}$ and for each policy $u \in U(i)$, $l_{ij}(u)$ gives the probability to reach state j from state i when applying control u.

A *policy* $\pi = \{u_0, u_1, \ldots\}$ is a sequence of control, where each u_k is a function from the space state to the controls, with $u_k(i) \in U(i)$ for each $k \geq 0$. Let denote by i_k the state at time k. Once a policy is fixed, the sequence of i_k becomes a Markov chain with transition probability:

$$\mathbb{P}(i_{k+1} = j | i_k = i) = l_{ij}(u_k) \tag{5.6}$$

$$\sum_{j \in S} l_{ij}(u) = 1 \text{ for all } u \in U(i) \tag{5.7}$$

Note that Equation (5.6) shows that the transition probability depends only on the policy chosen at the state, while Equation (5.7) ensures that the sum over all the states is equal to 1. *Stationary policies* are policies of the form $\pi = \{u, u, u, \ldots\}$. At any time, the same control are used for each state. Lastly, a cost function g represents the cost of a transition from state i to state j under control u noted $g(i, u, j) \in \mathbb{R}$.

In the SSPP, we assume there is one *absorbing* state 0, which is a cost-free termination state. More formerly, the Markov chain has a state $0 \in \mathcal{E}$ such that $l_{00}(u) = 1$ and $g(0, u, 0) = 0$ for any $u \in U(0)$. Given an initial state, the objective is to find a stationary policy which minimizes the expected cost to reach the termination state 0. The method can be extended to the case with several absorbing states.

We define the *cost-to-go* vector as the expected cost to reach absorbing states starting from any state, with respect to a policy $\pi = \{u_0, u_1, \ldots\}$ as:

$$J^{\pi}(i) = \lim_{N->\infty} \mathbb{E}\left[\sum_{k=0}^{N-1} g(i_{k+1}, u_k, i_k) \middle| i_0 = i \right] \tag{5.8}$$

The limit defining the cost-to-go vector J^{π} exists and is finite thanks to a set of assumptions explained in Chapter 2 of [17]. The optimal policy u^* is such that for each state $i \in \mathcal{E}$, the optimal cost-to-go vector associated to the optimal policy J^* respects the following system of equations:

$$\begin{array}{rcll} J^*(i) & = & \min_{u(i) \in U(i)} \sum_{j \in \mathcal{E}} l_{ij}(u) \Big(g(i, u, j) + J^*(j) \Big) & \text{for all } i \in \mathcal{E} \\ & = & \sum_{j \in \mathcal{E}} l_{ij}(u^*) \Big(g(i, u^*, j) + J^*(j) \Big) & \text{for all } i \in \mathcal{E} \end{array} \tag{5.9}$$

Using a classical dynamic programming (see Chapter 2 of [17]), it is possible to compute the optimal policy u^*.

How to apply the method to the vehicle balancing problem

The available information when a truck is asking the central monitor for a mission is the following:

– the number of vehicles parked at each station

– the position of each truck

– the number of vehicles carried by each truck

A state of the system should keep all the former information. The optimal policy to give to a driver would then be the station he should drive to. Note that the number of vehicles to load or unload at each stop is not to be decided as the objective is to bring the number of vehicles as closed to the station objective θ_i as possible.

Let $n \in \mathbb{Z}_+$ be the number of stations and $N \in \mathbb{Z}_+$ the number of vehicles. Even if there is only a unique truck, the number of states is huge, as a simple calculation shows. Assume that the capacity of the truck and the capacity of each station are equal to C, ($K = C$ and $C_i = C$ for all $i \in V$), the number of states with a total of $k \leq N$ vehicles parked is the number of compositions of $k + n$ with summands bounded in number ($n + 1$ summands) and size (lower or equal to $C + 1$) multiplied by the number of positions for the truck. Using the formula given in the note I.15 page 45 of Philippe Flajolet and Robert Sedgewick's book [42], we get that this number of compositions is equal to $[z^k]\Big(z\frac{1-z^{C+1}}{1-z}\Big)^{(n+1)}$, where $[z^k]$ means that we take the coefficient of z^k in the Taylor expansion of the formula given afterwards. So we have the following number of states:

$$\text{number of states} = n \times \sum_{k=0}^{N} [z^k]\Big(z\frac{1-z^{C+1}}{1-z}\Big)^{(n+1)}$$

However this formula is not helpful for easily computing the number of these states, but suits more asymptotic studies. Forgetting the capacity, it is well-known that the number of compositions with summands bounded in number only is equal to $\binom{k+n}{n}$. Multiplying this number by n, it gives an upper bound on the number of states such that k vehicles are parked. A lower bound can also easily be obtained as follows: choose the first $\lfloor \frac{n+k-1}{C+1} \rfloor$ points regularly over all the integers from 0 to $k + n$. Then, choose the others among the remaining points. The difference of two consecutive points is then lower or equal to $C + 1$. This difference gives the number of vehicles parked at each station, plus one. Following this method, no station can be assigned more than C vehicles. So the lower bound on the number of states such that k vehicles

are parked is then

$$\text{lower bound on the number of states} = n \times \sum_{k=0}^{N} \binom{k+n-\lfloor \frac{n+k-1}{C+1} \rfloor}{n-\lfloor \frac{n+k-1}{C+1} \rfloor}$$

The number of states increases exponentially. For instance, if the network has only 3 stations of capacity 5, and that there is a unique truck of capacity $K = 5$ and that 10 vehicles are present in the network, the number of states is 2163.

Let assume that the probability matrix to go from a state to another one is known. We define that a state such that the number of vehicles parked at each station is equal to its objective number, whatever the truck position and load, is an absorbing state. We correct the corresponding transition probability. This assumption is false in practice but enables to use the SSPP method. It was done as this method aims at reaching a situation such that all stations are at their target state θ_i as fast as possible. Otherwise, the system would not have any absorbing state. A similar method could be then used, adding a discount factor in the computation of the cost-to-go vector of Equation (5.8). The policy that would then be obtained would the optimal policy for maintaining the system near to its target states.

Locally unbalanced, but globally balanced

The huge number of states makes any implementation of the former method prohibitive. Indeed even for small networks, the number of states would explode. However we can reduce the number of states to consider. Indeed, knowing the situation at each station may be pointless. As the objective is to model a real shared transport system, any user that shows up at an empty station to rent a vehicle and sees a station on the other side of the street which is full would be able to cross the street to rent a vehicle. The same behavior holds for a user returning a vehicle at a station which is full. Moreover, knowing the exact number of vehicles at each station at each time may be useless. Indeed, if there are 2 or 3 vehicles at a station, it may not drastically change the action to take in a state, as the corresponding station will anyway be seen as suffering from a shortage of vehicles. The same note applies to the truck load.

It leads to consider *clusters* of stations instead of all stations, where clusters are built via distance consideration. Each cluster gathers stations that are nearby. Moreover, the exact number of vehicles in each cluster or on the vehicle can be "forgotten". Three cases are considered, depending on the average level of occupation per cluster. Three zones – or *colors* – are defined as follows:

– a cluster is said to be *blue* if its average level of occupation is below a given threshold; in practice, it means that vehicles need to be brought because there is a shortage of them
– a cluster is said to be *red* if its average level of occupation is above a given threshold; in practice, it means that vehicles need to be taken out from this cluster because there are too many of them
– otherwise a cluster is said to be *white*; in practice, it means that some stations may be full and other empty, but they compensate each other and in the cluster are available a number of vehicles that is near the sum of the objectives of its stations

This color system is also used for the trucks. The exact number of vehicles carried is neglected. The cluster partition and the color division enable to cut drastically the number of state to be considered. And in practice they seem relevant, as having a truck sent to balance stations that are less than 300 meters away would be nonsense.

From now on, a state is the color of each cluster, the color of the vehicle and its position. With 4 clusters, the number of considered states is then $3^4 \times 3 \times 4 = 942$, which makes the reduction relevant. The issue of clustering is not addressed here. The method we used is the Dunn fuzzy c-means algorithm [37], simplified by Bezdec [18]. The method which is implemented takes in input the number of clusters and build the number of requested clusters, gathering stations such that each station is closest to the centroid of the cluster it belongs to than to any other centroid.

We define the *target color* of a cluster as the color it would have if all the stations it represents are at their objective θ_i. We define as the absorbing states the states such that each cluster is at its target color. Assume that clusters 1 and 2 are located in residential areas gathering stations with high target levels, clusters 3 is a workplace area containing stations with low target levels and cluster 4 contains stations having both high and low target levels. If we want to find the optimal solution the goal would then be to reach a state such that the cluster colors are $(blue, blue, red, white)$. Such states would enable people to go from there home to their workplace. Another advantage of this method is that all the computation work would have been done off line. In real time, the order will be given to the vehicle directly. Once the driver is told the cluster he has to drive to, the exact logistic operations to execute at his destination are calculated thanks to the former two-step heuristic method (Subsection 5.2.2), computed only over the destination cluster stations.

Let define a distance between two colors as follows:

$$\text{Distance between } col_1 \text{ and } col_2 \ = \begin{cases} 0 & \text{if } col_1 = col_2; \\ 1 & \text{if } col_1 \neq col_2 \text{ and one of the two is white;} \\ 2 & \text{otherwise.} \end{cases}$$

The cost function g that is used here is the sum of all the distance between the cluster color of the current state and the one of the target state. First experiences have been done taking into account the distance driven by trucks, but it seems quite irrelevant: in real time, the objective is not to minimize the distance as trucks will drive through the city anyway, so the objective should not be to try to minimize the length of their journey but to maintain the regulation of the system. The color distance objective fits better the objective of service quality.

The probability matrix is assumed to be known in the former method. In our problem the evolution of the system is complex and we do not have it. However, thanks to the simulator outlined in Chapter 4, we can estimate them by running a huge number of experiences without trucks and stopping them after a given time which is an average time for a trip to occur in the city, and see the new repartition of vehicles. This enables us to evaluate the l_{ij} formerly mentioned, though the truck action is neglected.

5.3 Methods using incentive policy

In this section, the outlined method does not include the use of trucks. A price is attached to each station. These prices are updated regularly and aim at deterring users from parking at stations that are already nearly full. Users would have incentive to park to other stations that have a greater number of available parking places. They can yet follow their original demand and try to park at their original target station. But they would have to pay the price announced and are not sure to find an available parking place there. If they modify their trip and try to park at another station with a lower fee – meaning the station is quite empty – they are still not ensured to find an available parking place. However if prices are well-estimated, it should be quite unlikely.

The idea which is behind the heuristic is the resolution of Monge's transportation problem first introduced by Gaspard Monge in 1791 [61]. Assume users can decide to stop at a different vertex than their original target one. They would have to undergo an *extra cost* for walking from the intermediate vertex to their target one. Let have

$e_{ijk} := \mathbb{E}(\text{PRICEVHC} \times T^v_{ik} + \text{PRICEWALK} \times T^w_{kj} - \text{PRICEVHC} \times T^v_{ij})$, which models the *effort* provided by a user choosing an intermediate target vertex k on which he leaves his vehicle instead of going directly from i to j. PRICEVHC and PRICEWALK factors represent the prices users are willing to pay for using a vehicle (see Subsection 4.2.2 in the former chapter). Assume that to each station a price $t_k \in \mathbb{R}$ is associated. This cost has to be paid by any user parking his vehicle at station k. The total cost undergone by a user renting a vehicle at station i to go to station j but parking his vehicle at an intermediate target vertex k is then: $c_{ijk} = e_{ijk} + t_k$. Users behave rationally, so they choose for intermediate target vertex the one that minimizes the former cost. The objective here is to find the prices $t_k \in \mathbb{R}$ to associate to each station to maintain the number of vehicles at each station close to its target occupation level θ_i.

Let $x_{ijk} \geq 0$ be the number of users that want to go from vertex i to vertex j and choose station k as intermediate target vertex. We would like to have the following equations satisfied:

$$\begin{array}{lcll} \displaystyle\sum_{(i,j)\in V^2} x_{ijk} & = & T_k & \text{for all } k \in V \\ \displaystyle\sum_{k\in V} x_{ijk} & = & \lambda_i p_{ij} & \text{for all } (i,j) \in V^2 \\ x_{ijk} & \geq & 0 & \text{for } i,j,k \in V \text{ such that } k = \arg\min_{k'\in V} e_{ijk'} + t_{k'} \\ x_{ijk} & = & 0 & \text{for } i,j,k \in V \text{ such that } k \neq \arg\min_{k'\in V} e_{ijk'} + t_{k'} \end{array} \tag{5.10}$$

where T_k has to be proportional to $\max\{0, \theta_k - x_k\}$ is the number of vehicles to bring at station k per unit of time. The first equations of (5.10) ensure that the number of vehicles brought at any station matches this number, while the second one certify that the sum over all the stations is equal to the expected number of users to show up in the system.

Considering the following linear program

$$\begin{array}{lll} \min & \displaystyle\sum_{i\in V}\sum_{j\in V}\sum_{k\in V} e_{ijk}x_{ijk} & \\ \text{s.t.} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = T_k & \text{for } k \in V \\ & \displaystyle\sum_{k=1}^{n} x_{ijk} = \lambda_i p_{ij} & \text{for } i,j \in V \\ & x_{ijk} \geq 0 & \text{for } i,j,k \in V \end{array} \tag{5.11}$$

and its dual

$$\begin{array}{lll} \max & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i p_{uj}\omega_{ij} + \sum_{k\in V} T_k\mu_k & \\ \text{s.t.} & \mu_k + \omega_{ij} \leq e_{ijk} & \text{for } i,j,k \in V \end{array} \tag{5.12}$$

Proposition 5.3.1. *Let ω^*_{ij} et μ^*_k be the optimal solutions of the linear program* (5.12)*. Setting $t_k = -\mu^*_k$ for each vertex k ensures that Equation* (5.10) *has a solution.*

Proof. We want to prove that for such t_k's, program (5.10) has a solution. We will prove that actually the optimal solution x^*_{ijk} of program (5.11) is a solution of program (5.10).

The x^*_{ijk}'s satisfy all equations of the system (5.10), except maybe the last one.

Consider a pair $i, j \in V$. If $\lambda_i p_{ij} = 0$, then for all k we have $x^*_{ijk} = 0$, and all such x_{ijk}'s satisfy also the last equation. If $\lambda_i p_{ij} > 0$, then $x^*_{ijk} > 0$ for at least one k. By the complementary slackness, we have for such a k the equality $\mu^*_k + \omega^*_{ij} = e_{ijk}$. Hence $\omega^*_{ij} \leq e_{ijk} - \mu^*_k$ with equality when k is $\arg\min_{k' \in V}\{e_{ijk'} - \mu^*_{k'}\}$, i.e. $\omega^*_{ij} = \min_{k' \in V}\{e_{ijk'} - \mu^*_{k'}\}$. Now, take a k such that $e_{ijk} - \mu^*_k$ is not minimum. Then $\omega^*_{ij} < e_{ijk} - \mu^*_k$ and still by complementary slackness, we have $x^*_{ijk} = 0$.

Therefore, setting $t_k = -\mu^*_k$ for each vertex k makes x^*_{ijk} a solution of the system (5.10). □

Note that the value ω^*_{ij} in the proof above is the cost experienced by users showing up at station i and having for target station j when the prices are set to $-\mu^*_k$.

5.4 Computational study

5.4.1 Instances

In this section, the results that were obtained using all the former methods are given. The instances we used are available on the website of the project. Two cities were built – Edoras and Hyrule – using the CityBuilder tool mentioned in the Chapter 4. They were chosen for having a good repartition of the stations over the city space. The O-D matrix was built with respect to a gravity model. Four sizes were tested – with $20, 50, 100$ and 250 stations. Moreover, in all cases, three different types of demand are possible:

- In the "low demand case" a user is showing up at each station in average every 5 minutes
- In the "medium demand case" a user is showing up at each station in average every 2.5 minutes
- In the "high demand case" a user is showing up at each station in average every 1.5 minutes

The simulation time is set to four hours. For each case, a simulation with 10 replications using different randomly generated demand realizations are run. When the system starts, all the

vehicles are parked at stations. All the driving/walking/riding time from a station to another are deterministic. They all were run with one truck of capacity 20. As for the objective occupation level of stations θ_i, it is taken as 0.7 its capacity. For the Colored Cluster heuristic method, the threshold for the blue (respectively red) color is 0.2% (resp. 0.8%) of a cluster capacity. In the Pricing method, prices are updated every 15 minutes. Only one profile of user is present in the simulator with the following value for parameters: PRICEVHC$= 1$ and PRICEWALK$= 5$; users are willing to explore one station and spend 600 seconds maximum to find a vehicle, one station and 900 seconds maximum to find a parking place. If in a real system users would be willing to explore more stations, especially to find a parking place, these limits were knowingly taken low in order to be able to compare the results of the different methods. When the simulation starts, no vehicle are in use. The regulation methods start after 30 minutes to compare their performances on a running system.

5.4.2 Results

Tables 5.1, 5.2 and 5.3 respectively gather the results obtained running the methods on the Edoras instances. The figures correspond to the percentage of each category of users. A user is satisfied when he successfully rents a vehicle and returns it, within the limits of his patience if exploration occurs. The category "No vehicle" stands for users who did not succeed in renting a vehicle. The category "No parking" corresponds to the users who could not return their rent vehicle at their target station and did not find available parking place. Lastly, the "Rejection" category, which only exists in the case when the price method is used, is the percentage of users who chose to walk to their target station instead of paying a fee. The "Empty" columns show the results of the system without any regulation. "1SH" columns stand for the one-step heuristic method. "1SHF" columns give the results of the one-step heuristic with forecast method. "2SH" columns correspond to the results obtained by the two-step heuristic method. "2SHF" and "2S1SHF"columns respectively stand for the two-step heuristic with forecast method and the two-step one-stop heuristic with forecast method. Columns named "CC" show the results of the Colored Cluster heuristic method. The last columns show the performances of the pricing method – without trucks. Table 5.4 shows the ratio measuring the average number of satisfied users gained per vehicle moved (see Subsection 4.3.1 of the former chapter).

Size	Ind	Empty	1SH	1SHF	2SH	2SHF	2S1SHF	CC	Pricing
	Satisfied	70	99	99	98	95	98	72	84
20	No vehicle	12	1	1	0	3	1	12	0
	No parking	18	0	0	2	2	1	16	0
	Rejection								16
	Satisfied	80	92	94	90	88	92	83	91
50	No vehicle	13	5	4	5	9	5	11	0
	No parking	7	3	2	5	3	3	6	2
	Rejection								7
	Satisfied	62	67	67	66	66	67	65	81
100	No vehicle	26	23	23	23	23	22	23	10
	No parking	12	10	10	11	11	11	12	0
	Rejection								9
	Satisfied	63	65	65	65	65	65	66	
250	No vehicle	26	25	25	25	25	25	24	
	No parking	11	10	10	10	10	10	10	

Table 5.1: Comparison between performances of the different methods for Edoras in the low case demand

Size	Ind	Empty	1SH	1SHF	2SH	2SHF	2S1SHF	CC	Pricing
	Satisfied	59	91	93	85	82	88	59	80
20	No vehicle	31	7	5	10	15	10	31	4
	No parking	10	2	2	5	3	2	10	1
	Rejection								15
	Satisfied	75	84	86	82	81	83	78	86
50	No vehicle	19	13	11	14	15	14	17	7
	No parking	6	3	3	4	4	3	5	0
	Rejection								7
	Satisfied	50	54	55	54	54	54	51	69
100	No vehicle	41	38	37	38	38	38	40	22
	No parking	9	8	8	8	8	8	9	1
	Rejection								8
	Satisfied	46	47	47	47	47	47	48	
250	No vehicle	47	46	46	46	46	46	45	
	No parking	7	7	7	7	7	7	7	

Table 5.2: Comparison between performances of the different methods for Edoras in the medium case demand

Size	Ind	Empty	1SH	1SHF	2SH	2SHF	2S1SHF	CC	Pricing
	Satisfied	46	80	82	72	71	76	46	72
20	No vehicle	47	18	16	24	26	21	47	13
	No parking	7	2	2	4	3	3	7	1
	Rejection								14
	Satisfied	68	77	78	75	73	76	69	78
50	No vehicle	27	21	20	22	24	22	27	16
	No parking	5	2	2	3	3	2	4	0
	Rejection								6
	Satisfied	39	43	42	42	42	44	40	57
100	No vehicle	55	51	52	52	52	51	54	35
	No parking	6	6	6	6	6	5	6	1
	Rejection								7
	Satisfied	32	33	33	33	33	33	34	
250	No vehicle	63	63	63	63	63	63	62	
	No parking	5	4	4	4	4	4	4	

Table 5.3: Comparison between performances of the different methods for Edoras in the high case demand

Size	Demand	1SH	1SHF	2SH	2SHF	2SH1SF	CC
	Low	1.17	1.17	1.33	1.13	1.16	0.20
20	Medium	1.85	1.93	2.22	1.88	1.87	0.00
	High	3.11	3.22	3.58	3.30	3.08	1.44
	Low	1.23	1.24	1.38	1.03	1.27	0.37
50	Medium	1.80	2.20	1.94	1.83	1.77	0.81
	High	2.75	3.31	3.15	3.03	2.01	1.00
	Low	1.69	2.20	1.77	1.60	1.67	0.99
100	Medium	2.62	3.20	2.76	2.65	2.73	0.90
	High	4.00	4.08	4.07	3.78	3.90	0.73
	Low	2.31	2.92	2.37	1,89	2.28	1.87
250	Medium	1.50	4.30	3.35	3.11	3.17	2.71
	High	3.57	4.65	3.69	3.48	3.43	3.30

Table 5.4: Ratio users gained per vehicle moved for the different methods run on all Edoras instances

5.4.3 Discussion

The results show that with no doubt adding regulation improve the system level of service. The method using trucks that obtains the best results is the one-step heuristic with forecasts. Taking into account the forecasts improve the results in term of percentage of satisfied users whatever the demand for the instances with up to 100 stations. This could be done comparing the versions without and with forecast of the one-step and the two-step heuristic methods. Taking into account the future arrival of users improve the quality of the regulation. In term of the ratios, the values are slightly lower when taking into account the forecasts. The number of vehicles moved stays around the same value but the impact is better, as the value of the ratio decreases. The performances of the 2S1SHF is always better than the one of the 2SHF. The variation of the system being quick, using long planning horizon gives poorer performances as the second move may be done though the network differs strongly from the forecasts. Thus, short-term horizon, such as the one-step methods and the two-step one-stop heuristic with forecast method are more suitable. It is confirmed with the weak results of the CC method, though it is also due to the low quality of evaluation of probabilities of the transition in the Markov chain.

When the network is too big, one truck is not enough. Especially in the case of high demand. This enhances the idea of having the city divided into clusters with a reasonable number of stations in each of them, and having a truck devoted to balance operations in each one.

The method that seems to give the best results in term of user satisfaction is the pricing method. As for the prices, to make the conversion, we take EUR $8 = 1$ hour. It is a reasonable conversion for the value of travel time in a Western city (see [28]). The maximal magnitudes are about EUR 7 for 20 stations, EUR 10 for 50 stations and EUR 16 for 100 stations, these later values obtained only during a few minutes. In average, the magnitude is lower.

5.5 Conclusion

We have here presented different heuristic methods and implemented them in the simulator that was outlined in the previous chapter. Most of the method use trucks. One other proposes an incentive policy which updates prices to each station every 15 minutes. The methods are run on different instances with different sizes and demands. The results are gathered in tables and compared in the last section. The impact of forecasting the future state of the system is outlined. The diversity of the methods tested proves the versatility of the simulator OADLIBSim.

We can also make concrete recommendations.

- Simple short-term methods seems to be more efficient than long-term planning. This is due to the limited impact of the truck. It is true whatever the size or the demand of the network. As the system is evolving quickly, strategies should be evaluated continuously to correct them in the case a better action can be executed.
- Clustering the city could help. Having one truck regulating more than 50 stations diminishes clearly the performances that can be obtained.
- To enhance the regulation, having a good knowledge of the system enables to make accurate forecasts on the future state of the stations. This underlines the importance of having statistics on the system behaviour and to use them.
- The pricing strategy seems to be very promising with respect to its results. However it still needs to be addressed with socio-economical aspects.

Chapter 6

The Initial Inventory Problem

6.1 Introduction

In the former chapters, methods for real time repositioning were presented. They aim at improving the system efficiency by deploying vehicles from congested stations or areas of stations to others which are suffering from a shortage of vehicles. The two types of outlined methods use trucks to move vehicles or associate dynamic prices to stations to encourage users to self-regulate the network. Methods using trucks are possible and already in use in several BSSs. In Paris, Vélib' trucks are large enough to carry 20 bikes and the buses more than 60. Nevertheless, redeployment activities during the day participate in city congestion. This backfires on the efficiency objective as these trucks contribute to pollution when one of the objective of shared transport was to reduce greenhouse gas emission. Moreover, driving time are highly uncertain within the day, so the stations they are supposed to bring vehicles to can stay empty for a while. Furthermore, with a car sharing system such as Autolib, the limits of such operations appear clearly: having trucks driving around to move cars from stations to others is not likely to happen. Employees are brought to overloaded areas to move cars to areas suffering from shortage.

Conversely to day time, traffic in the city is nearly idle for a few hours overnight. Thus, repositioning activities can be achieved much more quicker. Night repositioning is indeed more efficient as trucks can balance twice more stations within an hour than they can during the day [21]. In the case of Autolib, a few hundreds of employees move cars from full stations to empty ones in the night. They started the system without planning on day repositioning activities [1], a choice that was modified after a few months. Most of the repositioning activities are yet still

done at night. Having night repositioning activities only presents another significant advantage that is trucks do not contribute to the day congestion. If the choice is made not to use trucks during day but only by night, a question is still to be addressed. If the users arrival rates per time slot at each station and the O-D matrix are known, what is the best quality of service which could be achieved? How many vehicles should be introduced in the system in order to minimize users' dissatisfaction? Raviv and Kolka [74] propose an algorithm to find the best initial inventory for one station in a BSS. Users who want to rent or park a vehicle arrive at each station according to non-homogeneous Poisson processes. When they cannot be served, they leave the system *unsatisfied*. The authors propose a convex function to model the total dissatisfaction with respect to the initial number of vehicles parked at a station. It enables to find the initial inventory that minimizes the users' dissatisfaction per station. In a shared transport network, users who cannot find a vehicle or a parking place at a station would try their luck in the neighboring stations, modifying the number of users arriving at the other stations. This report is not taken into account in their model. The idea of having most of the repositioning activities occurring at night is studied by Raviv *et al.* [76]. In their work, the dissatisfaction function introduced by Raviv and Kolka is used. One or several trucks are given a time window to achieve the best repositioning with respect to this users' dissatisfaction function. Several mathematical models are outlined and algorithm to solve them are presented.

The focus of this chapter is to find the best initial inventory at each station to get a better quality of service without using trucks during the day. This problem is called the *Initial Inventory Problem* (IIP). It is formally defined in Section 6.2, together with the way the performance of each initial inventory is measured. Section 6.3 outlines the method used to improve the performances of the system. Section 6.4 gives results that were obtained from realistic instances created thanks to data on an American city.

6.2 Initial Inventory Problem

As stated before, computing the evolution of a shared transport system with respect to independent users' decisions is a complex system. Thus, the simulator described in Chapter 4 is used to solve the IIP. The inputs are the same as those described in Section 4.2 of this chapter with the only difference that arrival processes of users are non-homogeneous Poisson: for each station, its mean arrival rate of clients per time slot is known and can change. The chosen time slot length is one hour. Simulation lengths are set to one day, starting at 12a.m. The O-D matrix

is homogeneous over time and gives the probability p_{ij} for a user arriving at station $i \in V$ to go to station $j \in V$, with $\sum_{j \in V} p_{ij} = 1$. The shared transport system here is seen as a mode of transport exclusively ($p_{ii} = 0$ for all $i \in V$).

In the IIP, the number of vehicles initially present in the network and their repartition have to be determined. The objective function to be minimized which measures the performance of an initial inventory is the total time lost by all users within the system because of inabilities to find a vehicle and/or a parking place. The simulator enables to compute the lost time as follows: when a user shows up, his departure station and target station are known. If everything runs perfectly for him, the *ideal time* he would spend within the simulator is the time for him to reach his target station with a vehicle leaving from his departure station. However, due to shortage of vehicles or parking places, it may happen that instead of starting from his departure station he starts from another one and/or that he parks at another station than his target one. These inconveniences extend the duration of his trip. This *extra time* can be measured by subtracting the real duration of his trip with the ideal one. The objective function is the sum over all the users of this extra time. Here, settings are such that no user abandons the system because of shortages of parking places or vehicles and they explore the system until they find either a vehicle or a parking place.

When users explore new stations to find a vehicle (respectively a parking place), the walking time (resp. time needed by the vehicle to go) from their current station to the station they choose to explore is added into the sum. In case a user arrives at his target station by walking, he leaves the system whether there is vehicles at his target station or not. When a user is expected to arrive at a station to rent or park a vehicle after the end of the simulation time, he is assumed to complete the rest of his trip without any further troubles. This assumption does not really impact the results presented in Section 6.4 as the instances that were used are realistic, and when simulations end, it is night time and very few users are actually using the system, with respect to the number of users over the day. The users' behaviour for choosing the stations to explore is the same as the one described in Chapter 4. At the end of the simulation, the ideal time is subtracted to the measured one to get the extra time

6.3 Finding the optimal initial inventory

In this section, two methods are outlined for solving the IIP. The idea of both methods is to run a large number of simulations with respect to an initial inventory and to check for each

station of the network if any *problem* occurs. By problem is meant that a user shows up and cannot find either a vehicle or a parking. The number of vehicles initially present at each station is then modified to try to prevent these users from being unsatisfied.

From now on, for each station, its *load curve* denotes the evolution of the number of vehicles parked at the station. Figure 6.1 illustrates the load curve at a station which capacity is 20. In the upper-line case, the station is full for about an hour around 6 p.m. If a user tries to return a rented vehicle during this time at the station, he would not succeed. If the initial number of vehicles initially present had been smaller, the station would not have been completely full, and such a user would park his vehicle successfully, as in the bottom-line case shows.

As the redeployment activity is supposed to be run once during the night, the only effect one can have is on the first problem that will occur at a station. Indeed, if a shortage of vehicles happens in the morning at a station and an excess in afternoon, adding or removing vehicles at the station cannot modify the excess to occur. Figures 6.2 and 6.3 show that in the case there is more than a few users unsatisfied during the first problematic period, the second periodic problem is unchanged. Modifying the initial number of vehicles does not have any impact on the evolution after the first problem happens, whether the successive problems are of the same type or not.

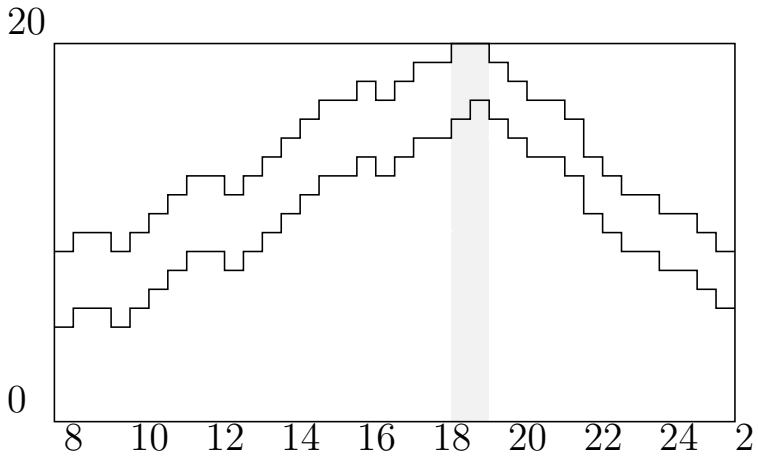

Figure 6.1: Load curve at a station with respect to its initial inventory

Both methods are local searches which modify the number of vehicles initially present. From an iteration to the next one, the initial number of vehicles at a station can vary by ± 1. Let us define two parameters $MaxChange \in \mathbb{Z}_+$ and $MaxIter \in \mathbb{Z}_+$. $MaxChange$ gives the maximum number of stations where the initial number of vehicles can be modified. This

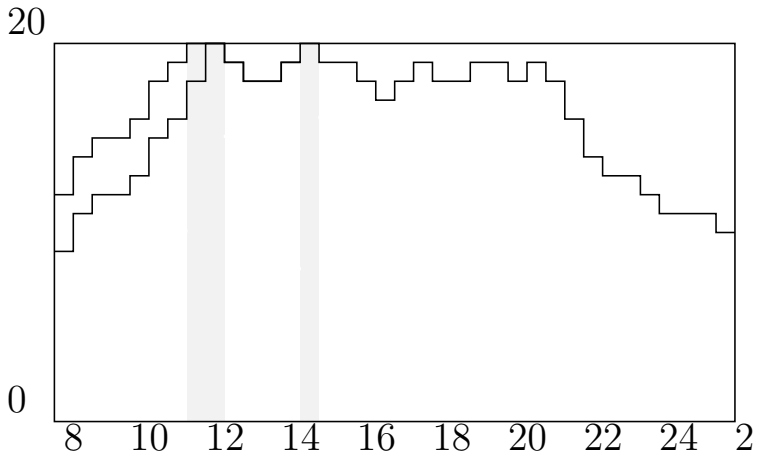

Figure 6.2: Example of how modifying the initial inventory could solve the first problem when a second same problem occurs

number is divided by two each time no improvement was found after $MaxIter$ consecutive iterations. The local search runs again starting from the best encountered solution. The expected gain that could be achieved by modifying the number of vehicles at a station can be roughly evaluated in the two methods. Stations are ranked with respect to this evaluation. When $MaxChange < n$ (n the total number of stations in the network), only the $MaxChange$ top ranked stations undergo changes.

6.3.1 Time driven search

Now it is settled that changing the number of vehicles initially parked at station can only influence the first problem. In this method, multiple simulations are run. When a problem occurs at a station which has never experienced one, the extra time caused by the problem and its type are kept. At the end of all the simulations, for each station, the extra time caused by shortage and excess problems over all the simulations are compared. If the former is greater (resp. lower) than the later, adding (resp. removing) a vehicle at the start of the simulation could help to reduce the total extra time of the simulation. The difference between these two values enables us to roughly measure the potential gain that could be achieved.

At the end, all the stations are sorted with respect to their potential gain. Note that if a station does not encounter any problem, this gain is 0 and so it is left aside of the former list. The first $MaxChange$ stations – if there is a sufficient number of them – see their initial number of vehicles modified by one when possible. Indeed, if for instance the station is initially

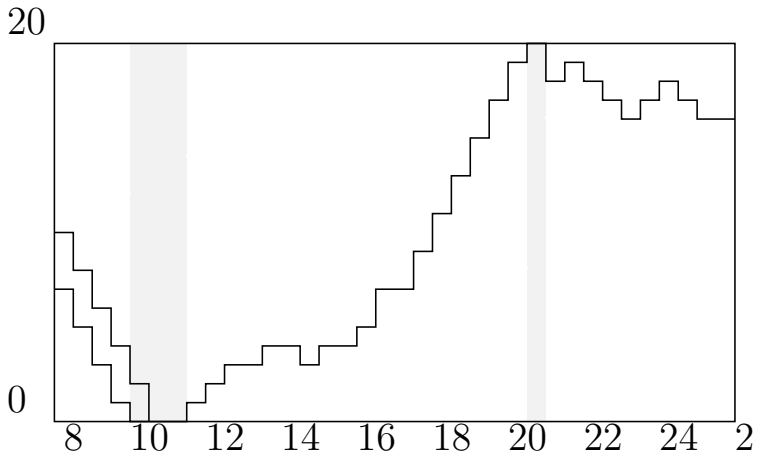

Figure 6.3: Example of how modifying the initial inventory could solve the first problem when a second different problem occurs

full, we cannot add any vehicle at it.

6.3.2 Occurrence driven search

In this method, the idea is not to focus on the extra time caused by a problem, but to study the load curve. Let assume that the load of a station goes down to zero before increasing again up to its capacity. If no user shows up to rent a vehicle while the station is empty, but at least one tries to park its rented vehicle when it is full, the problem that will first appear is the shortage of parking place. The former local search method could lead us to decrease the initial number of vehicles at the station. However one can understand that it will cause another "first problem" to occur, a shortage of vehicles, without solving the second problem anyway, as illustrated in Figure 6.4. The upper load curve is the original simulation. At about 11a.m., the station is empty and we assume no one arrives to rent a vehicle before one is returned to the station. At 9p.m., the station is full for about 30 minutes. Here we assume that at least one user tries to return his rented vehicle. Decreasing the initial number of vehicle by one would not impact on the fact the station is full in the evening. Moreover, it would cause a shortage problem at 11a.m., as shown by the fact the second line gets to null before the upper one.

To try to handle the situations when the load curve gets to zero or to the station capacity, six different types of *events* are distinguished. Running several simulations, the number of occurrences of each event for each station is kept in the following variables:

– $\#away$: the load curve stays away from both limits

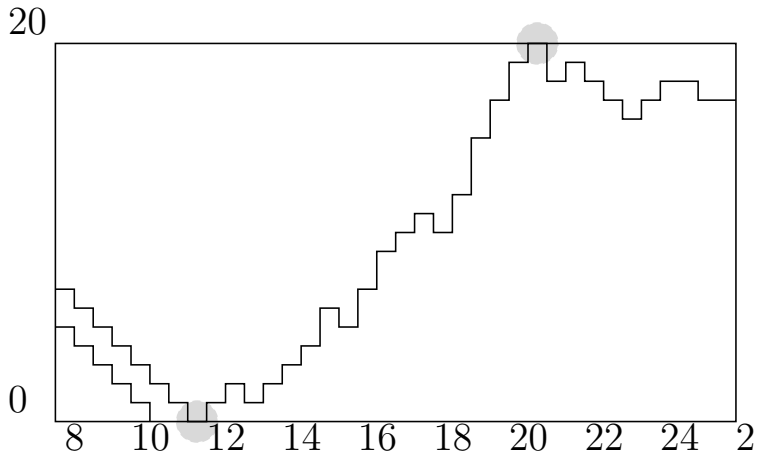

Figure 6.4: Decreasing the number of vehicles when the load curve gets to null

- $\#loadFactor_{up}$: the load curve reaches the station capacity without any problem to occur
- $\#excess$: the load curve reaches the station capacity and an excess of vehicles type of problem occurs
- $\#loadFactor_{down}$: the load curve is empty without any problem to occur
- $\#shortage$: the load curve is empty and a shortage of vehicles type of problem occurs
- $\#loadFactor_{both}$: the load curve bumps into both limits without any problem to occur

This discrimination between these events enables to roughly evaluate the consequences of adding or removing a vehicle in term of the number of occurrences of events as follows:

- Adding a vehicle at a station could cause a gain of

$$\#shortage - (\#excess + \#loadFactor_{up} + \#loadFactor_{both})$$

  Indeed $shortage$ events may decrease, but the $excess$ events will not. Moreover the $loadFactor_{up}$ and $loadFactor_{both}$ events would then become $excess$ events.
- Removing a vehicle could cause a gain of

$$\#excess - (\#shortage + \#loadFactor_{down} + \#loadFactor_{both})$$

  Indeed $excess$ events may decrease, but the $shortage$ events will not. Moreover the $loadFactor_{down}$ and $loadFactor_{both}$ events would then become $shortage$ events.

The $away$ events play no role in the evaluation. Modifying the number of vehicles initially present would either make them stay $away$ events or bump into one of the limits, turning them into $loadFactor_{down}$ or $loadFactor_{up}$ events. At the end of all the simulations, for each station,

the former gains over all the simulations are evaluated and sorted, and as previously the first $MaxChange$ operations are executed.

Although this method does not take into account the extra time, which is the objective function, the idea is that by minimizing the number of occurrences of problem events, the objective function value will decrease. Moreover this could help us to prevent from wrongly trying to solve problem such as illustrated by Figure 6.4.

6.4 Results

The experiments were done on modified data obtained from a big American city. The BSS has $n = 104$ stations, and a total of $3'084$ parking places for the vehicles. The simulation time is set to 24 hours, with new renters arrival rate at each station each hour. Instances were created in order to fit the problem features. All of them represent the same network. The differences come from the arrival rates. Arrival rates are computed from the one used in [74]. For each instance and for each station, a random value from 1 to 4 was drawn and its arrival rates over the day are multiplied by this value. In the simulations, in case of roaming, users' patience limits are set to 5 stations and 15 minutes, whether they are looking for a vehicle or a parking. This enables to have almost no clients leaving the system unsatisfied because of they could not find a vehicle or a parking place.

$MaxIter$ is set to 10 and $MaxChange$ is initially set to n. Table 6.1 sums up the results obtained by the both methods. For each case, a simulation with 100 replications using different randomwly generated demand realizations are run. Over the 100 replications, more than $30'000$ users show up in the system.

In table 6.1, the first column gives the number of the instance. The second column gives the initial repartition of vehicles used to run the search. Four different initial solutions are tested. The first one named "MIN_{SINGLE}" is the repartition obtained by Raviv and Kolka [74]. In the second case, the station are initially half-full. In the third case, they are all two-third-full. In the last case, the station initial number of vehicles is randomly chosen. The third and fourth columns show the initial number of vehicles and the initial amount of extra time. The fifth and sixth (respectively eight and ninth) columns give the final number of vehicles and mean extra time at the end of the time driven (resp. occurrence driven) search and the seventh (resp. tenth) columns show the mean extra time on 100 simulation changing the seed and starting from the initial inventory found by the time driven search (resp. occurrence driven search). All

the values are given in seconds.

Instance	Initial solution type	Initial Veh. number	Initial extra time	Time driven Veh. number	Time driven final time	Time driven test time	Occ. driven Veh. number	Occ. driven final time	Occ. driven test time
	MIN_{SINGLE}	1627	63547	1520	53875	54991	1566	55789	57119
Instance	$HALF$	1542	117722	1509	53974	54948	1565	55982	57524
0	$TWO-THIRD$	2056	201449	1626	54026	55328	1680	55838	57594
	$RANDOM$	1522	227498	1512	54039	55303	1566	56493	58054
	MIN_{SINGLE}	1588	132537	1488	117019	116030	1542	122891	121950
Instance	$HALF$	1542	231999	1508	117330	116131	1554	123335	122206
1	$TWO-THIRD$	2056	330863	1620	117096	115963	1660	123273	122051
	$RANDOM$	1517	276846	1485	116960	115818	1532	123412	122413
	MIN_{SINGLE}	1564	94384	1495	88432	88624	1548	90631	90989
Instance	$HALF$	1542	175725	1485	89163	89391	1545	91896	92205
2	$TWO-THIRD$	2056	282181	1584	88969	88944	1630	91838	92220
	$RANDOM$	1497	272356	1458	89478	89736	1519	92172	92711
	MIN_{SINGLE}	1611	87402	1527	76165	78905	1576	79757	82635
Instance	$HALF$	1542	172125	1563	76385	79373	1604	80347	83548
3	$TWO-THIRD$	2056	270476	1635	76233	79368	1687	80421	83373
	$RANDOM$	1640	250023	1533	76753	79908	1576	80234	83373
	MIN_{SINGLE}	1528	111551	1458	104669	104055	1510	107168	107248
Instance	$HALF$	1542	240496	1473	104571	103614	1527	107657	107754
4	$TWO-THIRD$	2056	342756	1566	104627	103931	1620	107579	107928
	$RANDOM$	1524	274125	1445	104576	103978	1505	107798	108064

Table 6.1: Performances of both local searches

Both searches lead to similar results but the time driven search gives better results than the occurrence driven one. In both cases, a local minimum is found. The search is robust as the objective function values are very close, whatever the initial solution. The initial inventory proposed by Raviv and Kolka algorithm is better than the other initial starts.

In each of the five instances, the initial inventories that are obtained are quite similar for most stations, with an initial number of vehicles varying around the same values. However for few stations, the difference can be higher. These stations are the major source of the variations in the time lost and the total number of vehicles initially displayed in the system. This could show that some stations are critical, while in others the initial inventories are less important. Figure 6.5 shows the values of the standard deviations measured on the computed initial inventories for Instance 0. The values goes from 0 to a little more than 7, and the mean standard deviation is equal to 1.19. We see that 85% of the stations have a standard deviation on their initial inventory that is lower than 2. Similar results are obtained looking at the other instances.

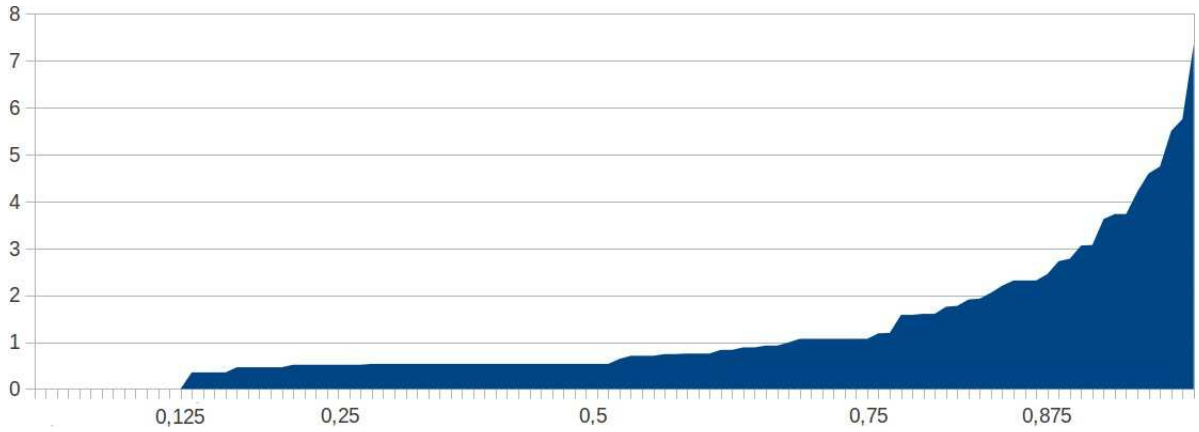

Figure 6.5: Repartition of the standard deviation on the computed initial inventories over the stations for Instance 0

6.5 Conclusion

In this chapter, a new problem is outlined. Chapters 2 and 3 addressed the issue of balancing a network with one or several trucks, taking as an input the initial number of vehicles to park at each station. Solving the IIP completes them by proposing a simple model to obtain a good initial repartition of the vehicles. The two local searches which are presented are quite simple and give satisfying results as the objective function value is divided by up to 4 when starting from random distribution or by 2 when starting from half-filled stations. When starting from the initial inventory obtained by Raviv and Kolka [74], the results can be improved to a certain extent only.

A trail for improving the search could be to combine both methods, changing the one used when local minima is found. Another work could be to try to have balancing operations twice a day – at night and in the beginning of the afternoon for instance. Running twice the method could help to reduce the value of the objective function and to enhance the system ability to answer users' demand. Last, the number of vehicle initially parked may differ by a few dozens without a real impact on the results. Another idea could be to find the best initial inventory using the smallest number of vehicles.

Once all vehicle renting and returning operations are known, an open question is to compute a lower bound on the extra time. Indeed, the lower bound used here – the ideal time – assumes that all trips can be satisfied. However if at a station of capacity 20, more than 20 users show up in a row without any vehicle being returned, whatever the initial inventory of the station, the last users cannot find any vehicles and they have to roam. Computing the value of this lower

bound could provide a more accurate estimation of the performance of the method.

Chapter 7

Conclusion

The object of this thesis is shared transport systems, with a focus on bikes sharing systems. The exploitation of such means of transportation induces several problems. In this thesis, imbalance problems are more specifically addressed from an OR point of view. The technical difficulties or vandalism are not treated. Imbalances are a tangible issue and it seems that improvements can be achieved without requiring huge investments. Various problems have been addressed: the Single-Vehicle One-Commodity Pickup and Delivery Problem (SVOCPDP), the Multiple-Vehicle Balancing Problem (MVBP), and the Initial Inventory Problem (IIP) which are related to night balancing activities mainly; to deal with the daytime issue, the simulator OADLIBSim was developed and several real-time methods were tested and compared. For each problem, exact algorithms and heuristics were outlined. Their performances were evaluated on a set of instances.

Chapter 2 is devoted to the SVOCPDP. A heuristic tabu search method is proposed to solve it. A branch-and-cut algorithm was presented for solving a relaxation on the initial problem. Although the relaxation is tight, building a real solution from an optimal solution of the relaxation is proven to be an $\mathcal{NP}$-hard problem. The overall algorithm is divided into three phases: at first, the tabu search is run starting from a greedy-made first solution. Then, the branch-and-cut algorithm runs with a time limit. At last, the best solution of the branch-and-cut algorithm is set up as a starting point of the tabu search. Several cuts are checked to enhance the relaxation. In most of the instances with up to 60 stations and a truck capacities over 20, the solution is less than 5% above the optimal solution.

For the MVBP of Chapter 3, a column generation approach is presented. Once the problem was defined and compared with other problems in the literature, a mathematical set partitioning-

like model is given. Because of an exponential number of variables, a column-and-cut generation method is proposed. The pricing subproblem is solved thanks to an enumeration enhanced by several dominance rules. A lower bound on the completion of half-paths is obtained with a dynamic programming algorithm. The linear relaxation is enhanced thanks to several cuts, some of them are dual feasible functions, others come from the dominances properties satisfied by optimal solutions. An memetic algorithm is used to obtain an upper bound on the original problem. The method is tested on instances with up to 40 stations and on the instances of the Split Delivery Vehicle Routing Problem (SDVRP), since the latter one is a special case of our problem when all the pickup activities are to be done at the depot. The optimal solutions are found for all 10 stations instances and some of the 20 stations instances. On the SDVRP instances, 5 of the 6 optimal solution found by Archetti *et al.* are obtained with our method.

To deal with real time balancing issues, Chapter 4 presents the simulator OADLIBSim. It enables to reproduce the evolution of a shared transport system. OADLIBSim offers the possibility to easily experiment regulation methods using trucks or affecting prices to each stations. Several indicators automatically computed measure the efficiency of different methods. In Chapter 5, a few of these heuristic methods are presented. Most of them use trucks. There is also a method using prices and based on duality and the Monge's transportation problem. They were all tested on a set of instances with different sizes and demand rates. The short-term methods seem to obtain better results than long-term ones. The method using the prices gets the best results, though it is the most complex to introduce in a real shared transport system.

For the IIP outlined in Chapter 6, OADLIBSim was used. Only night repositioning operations are done. The objective is to find the initial number of vehicles to put at each station in order to minimize the overall time lost by all users within the system. Modifying this number by one at a station could prevent a shortage or an excess of vehicles event to occur. We prove that such a shift in the number of vehicles may only impact the first "problem" which occurs within the day and propose two different local searches. They are then tested on instances that come from data obtained on an American city. The results show that the overall lost time can be divided by up to 4 starting from random initial repartitions but that it could even be reduced starting from the solution obtained by the algorithm of Raviv and Kolka which tries to minimize the number of unsatisfied users per station but does not take into account the roaming between stations.

Bibliography

[1] Autolib' dans la dernière ligne droite. *Le Parisien*, November 2011.

[2] Vélib' et moi, http://blog.velib.paris.fr/blog/2011/06/30/la-regulation-succes-et-ambitions, June 2011.

[3] Denver b-cycle, http://denver.bcycle.com, 2012.

[4] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[5] Shoshana Anily and Julien Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics (NRL)*, 46(6):654–670, 1999.

[6] Shoshana Anily and Refael Hassin. The Swapping problem. *Networks*, 22:419 – 433, 1992.

[7] Claudia Archetti, Maria Speranza, and Alain Hertz. A Tabu Search Algorithm for the Split Delivery Routing Problem. *Transportation Science*, 40, 2006.

[8] Claudia Archetti and Maria Grazia Speranza. An overview on the split delivery vehicle routing problem. In Karl-Heinz Waldmann and Ulrike M. Stocker, editors, *Operations Research Proceedings 2006*, Operations Research Proceedings, pages 123–127. Springer Berlin Heidelberg, 2007.

[9] Claudia Archetti, Nicola Bianchessi Maria Garzia Speranza, and Alain Hertz. A column generation approach for the split delivery vehicle routing problem. *NETWORKS*, 2011.

[10] Roberto Baldacci, Enrico Bartolini, and Aristide Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2):414–426, March/April 2011.

[11] Roberto Baldacci, Nicos Christophides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Math. Program.*, pages 351–385, 2008.

[12] Elsa Bambaron. Vélib' peine à trouver un second souffle. *Le Figaro*, Mars 2010.

[13] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46:316–329, 1996.

[14] Mike Benchimol, Pascal Benchimol, Benoît Chappert, Arnaud De La Taille, Fabien Laroche, Frédéric Meunier, and Ludovic Robinet. Balancing the stations of a self-service bike hire system. *RAIRO-Operations Research*, 45(1):37–61, January 2011.

[15] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 15(1):1–31, July 2007.

[16] Benoid Beroud. Les expériences de vélos en libre service en Europe. *Transports urbains*, 111, September 2007.

[17] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[18] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[19] B.Fr, Voigt, and Ilemerau. Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur. 1832.

[20] Charles Bordenave, Michel Gendreau, and Gilbert Laporte. A branch-and-cut algorithm for the preemptive swapping problem. *Networks*, 59(4):387–399, 2012.

[21] Villo! Brussels Bike Sharing System Operator. personnal communication, 2012.

[22] Jacques Carlier, François Clautiaux, and Aziz Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers and Operations Research*, 34:2223–2250, 2007.

[23] Daniel Chemla, Frédéric Meunier, and Roberto Wolfler Calvo. The multiple-vehicle balancing problem. *Working Paper*, 2012.

[24] Daniel Chemla, Frédéric Meunier, and Roberto Wolfler Calvo. Bike hiring system: solving the rebalancing problem in the static case. *Discrete Optimization*, in revision.

[25] Daniel Chemla, Frédéric Meunier, Thomas Pradeau, Roberto Wolfler Calvo, and Houssame Yahiaoui. Bike sharing system: simulation, repositioning, pricing. *Working Paper*, 2012.

[26] François Clautiaux, Claudia Alves, and José Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179:317–342, 2010.

[27] Claudio Contardo, Catherine Morency, and Louis-Martin Rousseau. Balancing a Dynamic Public Bike-Sharing System. Technical report, March 2012.

[28] Yves Crozet. Le temps et le transport de voyageurs. Technical report, European Conference of Ministers of Transport, 2005.

[29] George B. Dantzig, Ray Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[30] George B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[31] Paul deMaio. Bike-sharing: History, Impacts, Models of Provision, and Future. *Journal of Public Transportation*, page 41–56, 2009.

[32] Guy Desaulniers, Jacques Desrosiers, and Marius M.Solomon; GERAD. *Column Generation*. Springer, 2005.

[33] Martin Desrochers and Gilbert Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10:27–36, 1991.

[34] Moshe Dror and Pierre Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–145, May 1989.

[35] Moshe Dror and Pierre Trudeau. Split delivery routing. *Naval Research Logistics*, 37:383–402, 1990.

[36] Louis Dubost, Robert Gonzalez, and Claude Lemaréchal. A primal-proximal heuristic applied to the French unit-commitment problem. *Math. Program.*, 104(1):129–151, September 2005.

[37] Joseph C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[38] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[39] Pierrick Fay. Vélib', un système qui peut coûter cher. *Europe 1*, Avril 2011.

[40] Sándor P. Fekete and Jörg Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31, 2001.

[41] Erlon C. Finardi, Edson L. da Silva, and Claudia Sagastizàbal. Solving the unit commitment problem of hydropower plants via lagrangian relaxation and sequential quadratic programming. *Computational & Applied Mathematics*, 24:317 – 342, 12 2005.

[42] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[43] Antonio Frangioni, Claudio Gentile, and Fabrizio Lacalandra. Solving unit commitment problems with general ramp constraints. *International Journal of Electrical Power; Energy Systems*, 30(5):316 – 326, 2008.

[44] Christine Fricker and Nicolas Gast. Incentives and regulations in bike-sharing systems with stations of finite capacity. *Working Paper*, 2012.

[45] Tibor Gallai. Maximum-minimum theorems for networks. Technical report, 1957.

[46] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276 – 1290, 1994.

[47] Michel Gendreau, Gilbert Laporte, and Charles Bordenave. A branch-and-cut algorithm for the non-preemptive swapping problem. *Naval Research Logistics*, 2009.

[48] David K. George and Cathy H. Xia. Fleet-sizing and service availability for a vehicle rental system via closed queueing networks. *European Journal of Operational Research*, 211(1):198 – 207, 2011.

[49] Fred W. Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic, 1997.

[50] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10:196–210, March 1962.

[51] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. *Columns Generation*, pages 33–65, 2005.

[52] Josef Kallrath, Panos M.Pardalos, Steffen Rebennack, and Michel Scheidt. *Optimization in the Energy Industry*. 2009.

[53] Spiridon A. Kazarlis, Anastasios Bakirtzis, and Vassilios Petridis. A genetic algorithm approach to solve the unit commitment problem. *IEEE Transactions on Power Systems*, 11:83 – 92, February 1996.

[54] Mohand I. Khemmoudj, Marc Porcheron, and Hachémi Bennaceur. When constraints programming and local search solve the scheduling problem of edf nuclear power plant aoutages. *12th Internationnal Conference on Principles and Practice of Constraint Programming*, pages 271–283, 2006.

[55] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2nd edition, 2002.

[56] Neal Lathia, Saniul Ahmad, and Licia Capra. Measuring the impact of opening the london shared bicycle scheme to casual users. In *Transportation Research Part C*, 2012.

[57] Jenn-Rong Lin and Ta-Hui Yang. Strategic design of public bicycle sharing systems with service level constraints. *Transportation Research Part E: Logistics and Transportation Review*, 47(2):284 – 294, 2011.

[58] Marco E. Lubbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, pages 1007–1023, November-December 2005.

[59] Richard Lusby, Laurent Flindt Muller, and Bjorn Petersen. A solution approach to the roadef/euro 2010 challenge based on benders decomposition. Technical report, November 2010.

[60] Aristide Mingozzi. personnal communication, 2010.

[61] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. page 666–704, 1781.

[62] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms, 1989.

[63] Rajeev Motwani and Prasad Chalasani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28, 2009.

[64] Denis Naddef, Philippe Augerat, José M. Belenguer, Enrique Benavent, and Angel Corbéran. Separating capacity constraints in the cvrp using tabu search. *European Journal of Operational Research*, 106(2–3):546 – 557, 1998.

[65] United Nations Department of Economic and Social Affairs/Population Division. World urbanization prospects: The 2009 revision, 2010.

[66] Hipólito Hernandez Pérez and Juan-José Salazar González. The one-commodity pickup-and-delivery travelling salesman problem. *Lecture Notes in Computer Science*, 2570:89 – 104, 2002.

[67] Hipólito Hernandez Pérez and Juan-José Salazar González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145:126 – 139, 2004.

[68] Hipólito Hernandez Pérez and Juan-José Salazar González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38:245–255, 2004.

[69] Hipólito Hernandez Pérez and Juan-José Salazar González. The one-commodity pickup-and-delivery travelling salesman problem: inequalities and algorithms. *Networks*, 4:258–272, 2007.

[70] Hipólito Hernandez Pérez, Inmaculada Rodriguez Martin, and Juan-José Salazar González. A hybrid grasp/vnd heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers and Operations Research*, 36:1639–1645, 2009.

[71] Dana Pesach, Tal Raviv, and Michal Tzur. Dynamic Repositioning in a Bike-Sharing System Models and Solution Approaches. *Working Paper*, 2011.

[72] Marc Porcheron, Agnès Gorge, Olivier Juen, Tomas Simovic, and Guillaume Dereu ; EDF R&D. Challenge roadef/euro 2010 :a large-scale energy management problem with varied constraints. February 2010.

[73] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985 – 2002, 2004.

[74] Tal Raviv and Ofer Kolka. Optimal inventory management of a bike-sharing station. *Working Paper*, 2011.

[75] Tal Raviv and Michal Tzur. personnal communication, 2011.

[76] Tal Raviv, Michal Tzur, and Iris Forma. Static repositioning in a bike-sharing system: Models and solution approaches. *ODYSSEUS IV*, 2009.

[77] R.M. A la recherche désespérée de villo. *La Libre*, September 2010.

[78] Antoine Rozenknop, Roberto Wolfler Calvo, Laurent Alfandari, Daniel Chemla, and Lucas Létocart. Solving eletricity production planning by column generation. *Journal of Scheduling*, 2012.

[79] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[80] Susan A. Shaheen, Stacey Guzman, and Hua Zhang. Bikesharing in Europe, the Americas, and Asia: Past, Present, and Future. *Transportation Research Record*, 2143:159 – 167, 2010.

[81] Herbert A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, December 1962.

[82] Laura Spinney. Les maths au secours du "vélopartage". *Le Temps*, May 2011.

[83] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. 2002.

[84] Patrick Vogel and Dirk C. Mattfeld. Anticipating usage patterns in the design of bike-sharing systems. In *INFORMS 2011; Shared Mobility Systems*, 2011.

[85] Albert Wagelmans, Stan van Hoesel, and Antoon Kolen. Economic lot sizing: An o(n log n) algorithm that runs in linear time in the wagner-whitin case. *Operations Research*, 40:145–156, January-February 1992.

[86] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89 – 96, October 1958.

[87] Laurence A. Wolsey. *Integer Programming*. 1998.

[88] Konstantinos Zavitsas, Ioannis Kaparias, and Michael G.H. Bell. Transport problems in cities. Technical report, 2010.

Part III

Appendix

This part gathers the work done in the two ROADEF challenges.

Chapter A is the article [78] published following the 2010 edition presenting the work done with Roberto Wolfler Calvo, Antoine Rozenknop, Laurent Alfandari and Lucas Létocart.

Chapter B presents briefly the method used to solve the 2012 edition. This work was done with Bernat Gacias and Paolo Gianessi.

Appendix A

Challenge ROADEF 2010: Solving electricity production planning by column generation

This work has been done together with Antoine Rozenknop, Roberto Wolfler Calvo, Laurent Alfandari and Lucas Létocart.

A.1 Introduction

Optimization in the energy sector is a large subject. With the raise of raw materials prices and the growing awareness for environment concerns, energy consumption must be considered in a less greedy way in order to save resources. For years, several researchers have worked on optimizing the energy industry. In [52], the authors gather such type of problems, one of which is the electricity production planning. Electricity can be produced through different ways: nuclear and other thermal energies are the major part of production, but hydraulic energy or other renewable energies have sometimes a significant role (in Norway almost all the production is done using hydraulic energy). Producers can mix these different production resources in order to satisfy the energy demand while decreasing their costs and/or their impact on the environment. Electricity companies look for effective optimization tools since even small percentages of gain through optimization translate into large savings. The problem of optimizing energy production planning has several difficulties, one of which is that the exact amount of the future demand and the exact price of raw materials are unknown. This difficulty naturally increases

with the length of the time horizon, which can be in some cases very large.

A typical energy production planning problem with a short-term horizon is the Unit Commitment Problem (UCP). In the UCP, several generators are available for producing energy. Costs and constraints are associated with generators depending on their type: nuclear and other thermal units, hydraulic energy, wind turbines. For example, in a short-term horizon, the level of water in dams has to stay above some threshold. A forecast of the demand over a period of time is given. The objective is to find the best production assignment in order to satisfy the demand while minimizing production costs. Kazarlis *et al.* [53] propose a genetic algorithm to solve the UCP. Claudia Sagastizàbal *et al.* [41] present a Lagrangian relaxation method. Frangioni *et al.* in [43] extend Lagrangian relaxation methods to the UCP with ramping constraints, i.e., constraints that bound the difference of production of thermal units from one time step to the next. Claude Lemaréchal *et al.* [36] also studied UCP in the French context.

In long-term planning, typical constraints on energy production relate to maintenance operations that have to be executed periodically on nuclear plants. No production can be planned during these maintenance periods. The problem of scheduling the shut down of nuclear power plants for refueling and maintenance (see for example [54]) has been proposed as subject of the EURO/ROADEF 2010 challenge. EDF power generation facilities in France stand for a total of 98.8 GW of installed capacity and mix thermal energy (90 % in 2008, among which 86 % is produced by nuclear power and the rest by coal, fuel oil and gas), hydraulic and other renewable energies. The research project consists in modeling the electricity production assets, finding an optimal schedule of outages for each nuclear plant over a given time horizon, and determining an optimal production plan to satisfy demand. Moreover, in long-term electricity production planning, numerous uncertainties have to be taken into account, such as demand variation, generation units availability, spot market prices, quantities that can be bought or sold. This uncertainty leads to consider multiple scenarios and asks for a robust outage schedule that has to be feasible for every scenario. The scheduling of nuclear plant outages has to comply with various safety constraints and limitations on resources which are necessary to perform refueling and maintenance operations. The production planning fixes the quantity of energy to be produced by each plant at each time step for each scenario, and is also subject to technical constraints. The objective of the problem is to minimize the expected production cost over all scenarios. The proposed approach is based on mathematical programming and problem decomposition. An other approach used for solving the challenge ROADEF, based on a different decomposition method has been proposed by Lubsy *et al.* [59].

In this paper a column generation method (see [32] and [58] for surveys on this approach) is proposed to solve it. Column generation is often used to solve large-scale optimization problems when the problem can be formulated as a mixed integer linear program with a huge number of variables, or *columns* – called the *master problem* – as it is the case in the EDF problem. Creating and keeping all variables is not possible and useless as most of them would not enter the optimal solution. A *restricted master problem* is then defined, where instead of solving the problem on all its solution space, only a subset of variables (or columns) is considered. The column generation method iterates between two different problems: the restricted master problem and a *pricing subproblem*. The pricing subproblem aims at looking for new columns with negative reduced cost to add to the restricted master problem to improve its optimal value.

Even if the set of columns of the original master problem has a huge or exponential size, it is often possible to find the minimum reduced cost column in short or polynomial time, depending on the structure of the pricing problem. The method stops when no more negative reduced cost column is produced by the pricing subproblem, which means that the linear relaxation of the original master problem is solved to optimality. Nevertheless, in the case of integer variables the optimal solution of the linear relaxation could be far away from the optimal solution of the original master problem. In this case it is necessary either to start a branch-and-price algorithm (see for example [13] for a survey on this topic) or to have a complete pricing of the whole set of columns candidate to be in the optimal solution (see for example [11] for a reference on this topic).

Our approach is a heuristic method since only a subset of the whole set of possible columns are implicitly considered, due to the time limit imposed by the competition. This method is indeed heuristic for several reasons. The first reason is that production levels are discretized over time periods. This implies that the pricing subproblem is approximately solved using a shortest path algorithm in a graph with a finite number of nodes associated with production levels, which is much faster than solving it with continuous variables. The second reason is that scenarios are aggregated into a single mean scenario. Finally, the column generation method leads to find the optimal solution of the relaxed problem on the subset of columns that could be produced with respect to the discretization considered. A branch-and-price algorithm was not envisaged due to the time limits. At the end of the column generation process, once the outages have been scheduled, two linear programming problems are solved to optimality for deciding the quantity of fuel and the optimal production planning over all scenarios for this set of outages.

The main contributions of this paper are the following:

– an original set partitioning-like reformulation of the problem;
– a pricing subproblem solved by shortest path computations on a particular state graph;
– a diversification method that changes the objective function, adding columns that may have a positive reduced cost;
– two linear programming problems solved to optimality for deciding the quantity of fuel and the production planning for the selected set of outages.

The discriminant strategy chosen for our approach is to spend time to generate a single high-quality schedule of outages, guided by the cost objective of column generation, rather than enumerating several schedules of outages in a shorter time.

This paper is organized as follow. The electricity production planning problem is presented in Section A.2. Section A.3 introduces the mathematical model that formulates the problem. Section A.4 explains the decomposition applied and describes the overall column generation method used to solve it. Section A.5 describes the formulation of the pricing subproblem and its resolution method. Section A.6 gives the model of the final master problem that is solved for fixing dates of outages and critical stock-level periods, ending the first stage of the method. Section A.7 shows how to improve the pricing subproblems in order to generate a greater number of columns with higher diversity. Section A.8 presents the second-stage process used for fixing reload and production quantities for each plant. Section A.9 gathers the numerical results obtained on the test instances while Section A.10 concludes the paper.

A.2 Problem statement

The notations used in this paper mix the notations used in the ROADEF/EURO 2010 Challenge document [72] and notations issued from the paper by Lubsy *et al.* [59]. The various indexes used are scenarios $s \in S$, time steps $t \in T_{=}\{1, \ldots, |T|\}$, weeks $w \in W = \{1, \ldots, |W|\}$, non-nuclear plants $j \in J$, nuclear plants $i \in I$ and production cycles $k \in K_i = \{1, \ldots, |K_i|\}$ for each nuclear power plant i, $|K_i|$ being the maximum number of cycles that can start during the considered period. The decision variables introduced in the Challenge document are:

– ha_{ik} : week of cycle k during which nuclear plant i goes offline (i.e. week of decoupling)
– p_{jts} : production of non-nuclear plant j during time step t over scenario s
– p_{its} : production of nuclear plant i during time step t over scenario s
– r_{ik} : reload performed during the outage of cycle k of nuclear plant i

– x_{its} : stock of fuel of nuclear power plant i at the beginning of time step t over scenario s

Variables ha are integer, all other variables are continuous variables.

The objective function is:

$$\min \sum_{i \in I} \sum_{k \in K} C_{ik} r_{ik} + \frac{1}{|S|} \sum_{s \in S} \left(\sum_{t \in T} \sum_{j \in J} C_{jts} p_{jts} D^t - \sum_{i \in I} C_i^f x_{is}^f \right)$$

where C_{ik} is the proportional cost of fuel during cycle k for nuclear plant i, C_{jts} is the proportional production cost for time t and scenario s for non-nuclear plant j, C_i^f is the selling price (or the gain) for the remaining fuel x_{is}^f and D^t is the length of time step t.

The set of constraints is the following, taking the same numbering as in the Challenge document. Some notations used throughout the paper are introduced with these constraints, amongst which :

– $T_{\mathcal{W}}$ is the set of time steps contained in a set of weeks $\mathcal{W}$,

– L_{ik} is the length of an outage (defined in constraint `[CT13]` but used before).

`[CT1]` Demand satisfaction:

$$\sum_{j \in J} p_{jts} + \sum_{i \in I} p_{its} = DEM_{t,s} \quad \forall s, t$$

`[CT2]` For non-nuclear plants, we only have minimum and maximum production constraints:

$$\underline{P}_{jts} \leq p_{jts} \leq \overline{P}_{jts} \quad \forall t, s, j$$

where $\underline{P}_{jts}$ and $\overline{P}_{jts}$ are the minimum and maximum production for non-nuclear plants j on time step t for scenario s.

`[CT3]` Offline power: if the nuclear power plant is offline the production must be 0

$$t \in T_{[ha_{ik}, ha_{ik}+L_{ik}[} \Rightarrow p_{its} = 0 \quad \forall t, s, j$$

`[CT4]` Minimum power: if the nuclear power plant is online the production must be greater or equal than 0

$$p_{its} \geq 0 \quad \forall i, t, s$$

`[CT5]` Maximum power before activation of imposition of power profile constraint

$$\text{if} \quad (t \in T_{[ha_{ik}+L_{ik}, ha_{i(k+1)}[}) \text{ and } x_{its} \geq BO_{ik} \Rightarrow p_{its} \leq \overline{P}_{it} \quad \forall i, t, s, k$$

where BO_{ik} is a bound on stock of fuel that activates the imposition of power profile during cycle k, and $\overline{P}_{it}$ is the maximum power of plant i during time step t

[CT6] Maximum power after activation of imposition of power level constraint :

$$\left.\begin{array}{l}\text{if} \quad t \in T_{[ha_{ik}+L_{ik},ha_{i(k+1)}[} \text{ and } x_{its} < BO_{ik} \text{ then} \\ \left\{\begin{array}{ll}\text{if} & x_{its} \geq (PB_{ik}(x_{its}) \times \overline{P}_{it}) \times D^t \\ \text{then} & \frac{p_{its}}{PB_{ik}(x_{its}) \times \overline{P}_{it}} \in [1-\epsilon, 1+\epsilon] \\ \text{else} & p_{its} = 0\end{array}\right.\end{array}\right\} \quad \forall s,t,i,k,$$

where $PB_{ik}(x_{its})$ is an imposed decreasing production profile during the production campaign of cycle k (piecewise linear function dependent on the level of fuel), and ϵ is a tolerance parameter. It is important to point out that these constraints are non linear.

[CT7] Bounds on refueling for scheduled outages k:

$$\underline{R}_{ik} \leq r_{ik} \leq \overline{R}_{ik} \quad \forall i,k$$

where $\underline{R}_{ik}$ and $\overline{R}_{ik}$ are the minimum and maximum quantity that can be refueled, respectively.

[CT8] Initial fuel stock :

$$x_{i0s} = XI_i \quad \forall (i,s)$$

[CT9] Fuel stock variation during a production campaign

$$t \in T_{[ha_{ik}+L_{ik},ha_{i(k+1)}[} \Rightarrow x_{i(t+1)s} = x_{its} - p_{its} \times D^t \quad \forall s,t,i,k,$$

[CT10] Fuel stock variation during an outage. If t is the first time step of ha_{ik} then

$$x_{i(t+1)s} = ((Q_{ik}-1)/Q_{ik})(x_{its} - BO_{i(k-1)}) + r_{ik} + BO_{ik} \quad \forall i,t,s,k$$

where Q_{ik} is a refueling coefficient during the outage of cycle k

[CT11] Fuel stock variation during an outage

$$t \text{ is the first time step of } ha_{ik} \Rightarrow x_{its} \leq A_{ik},\ x_{i(t+1)s} \leq S_{ik} \quad \forall s,t,i,k,$$

where A_{ik} (resp. S_{ik}) is the maximum bound on stock of fuel at the time of outage (resp. during production campaign) of cycle k

[CT12] Constraint on maximum modulation over a cycle

$$\sum_{t \in T_{[ha_{ik}+L_{ik},ha_{i(k+1)}[} \text{ and } x_{its} \geq BO_{ik}} (\overline{P}_{it} - p_{its}).D^t \leq \bar{M}_{ik} \quad \forall s,i,k$$

where $\bar{M}_{ik}$ is the maximum modulation over the production campaign of cycle k. These constraints state that the gap between total production and production capacity, during a production campaign staying above the BO stock threshold, should not be too high.

[CT13] Earliest and latest date of outage

$$ha_{i(k+1)} \geq ha_{ik} + L_{ik} \quad \forall i, k$$

$$T^o_{ik} \leq ha_{ik} \leq T^a_{ik} \quad \forall i, k$$

where L_{ik} is the number of weeks (length) of outage k. These constraints bound the earliest and the latest date of an outage for each cycle $k \in K_i$ of each plant i.

[CT13bis] No outage $k \in K_i$ could be skipped as soon as it has a last possible week $T^a_{i,k}$ defined.

In the next series of constraints: N^{xx}_m are number of weeks associated to constraint $m \in M^{xx}$ and Q^{xx}_m represent quantities.

[CT14] Minimum spacing and maximum overlapping between outages of i and i'

$$ha_{ik} - ha_{i'k'} - L_{i'k'} \geq N^{14}_m \text{ or } ha_{i'k'} - ha_{ik} - L_{ik} \geq N^{14}_m \quad \forall (i,k),(i',k') \in G^{14}_m, \forall m \in M^{14}$$

[CT15] Minimum spacing and maximum overlapping between outages during a specific period. Let's define with $[b_m, e_m]$ a time interval, then

if

$$(b_m - L_{ik} + 1 \leq ha_{ik} \leq e_m) \text{ and } (b_m - L_{i'k'} + 1 \leq ha_{i'k'} \leq e_m)$$

then

$$ha_{ik} - ha_{i'k'} - L_{i'k'} \geq N^{15}_m \text{ or } ha_{i'k'} - ha_{ik} - L_{ik} \geq N^{15}_m \quad \forall (i,k),(i',k') \in G^{15}_m, \forall m \in M^{15}$$

[CT16] Minimum spacing between decoupling dates

$$|ha_{ik} - ha_{i'k'}| \geq N^{16}_m \quad \forall (i,k),(i',k') \in G^{16}_m, \forall m \in M^{16}$$

[CT17] Minimum spacing between coupling dates

$$|ha_{ik} + L_{ik} - ha_{i'k'} - L_{i'k'}| \geq N^{17}_m \quad \forall (i,k),(i',k') \in G^{17}_m, \forall m \in M^{17}$$

[CT18] Minimum spacing between coupling and decoupling dates

$$|ha_{ik} + L_{ik} - ha_{i'k'}| \geq N^{18}_m \quad \forall (i,k),(i',k') \in G^{18}_m, \forall m \in M^{18}$$

[CT19] Resource constraints on outages. Let define with

– w_{ik}^m represents the start of using the resources for the outage, expressed in terms of number of weeks after the beginning of the outage.
– i_{ik}^m the number of weeks during with the resources are not available for the next outage
– Q_m^{19} quantity of available resources

$$\sum_{(i,k)\in G_m^{19}} \delta(t,i,k) \leq Q_m^{19} \quad \forall t \in T \quad \forall m \in M^{19}$$

where $\delta(t,i,k) = 1$ if $ha_{ik} + w_{ik}^m \leq t \leq ha_{ik} + w_{ik}^m + i_{ik}^m$

[CT20] Maximum number of overlapping outages during a specific week w_m

$$\sum_{(i,k)\in G_m^{20}} \delta(w_m,i,k) \leq Q_m^{20} \quad \forall m \in M^{20}$$

where Q_m^{20} is the maximum number of outages at week w_m and $\delta(w,i,k) = 1$ if $ha_{ik} \leq w < ha_{ik} + L_{ik}$

[CT21] Maximum offline power capacity of a set of power plants during a time period. For a given time period $[b_m, e_m]$ the power capacity of a subset of plants C_m^{21} that are offline is bounded by a maximal quantity Q_m^{21}

$$\sum_{i\in C_m^{21}} \sum_k \delta(w,i,k)\overline{P}_{it} \leq Q_m^{21} \quad \forall m \in M^{21},\ \forall w \in [b_m, e_m],\ \forall t \in T_w$$

A.3 Compact formulation

The solution approach is based on mathematical programming. Our compact formulation is inspired from [59]. However, we provide a *complete* compact formulation as some constraints ([CT6] and [CT12]) were not directly modeled in [59]. Before defining the Mixed Integer Programming (MIP) model, we introduce some additional notations, sets, constants and variables.

Sets and data

– $w(t)$: week containing time step t
– T_w : set of timesteps in week w
– $q_{ik} = \frac{Q_{ik}-1}{Q_{ik}}$ the ratio of fuel kept from a cycle to the next one

Variables

Variables r_{ik}, p_{jts} are the same as defined before, but new binary variables are introduced and variables x_{its} are splitted as described below.

- y_{iwk} (binary): $= 1$ if cycle k for plant i begins in week $w \in W$, 0 otherwise,
- Y_{ik} (binary): $= 1$ if cycle k is used for plant i, 0 otherwise,
- x^1_{iks} : fuel stock of nuclear power plant i at the beginning of cycle k (i.e. before refueling) over scenario s
- x^2_{iks} : fuel stock of nuclear power plant i after refueling at cycle k over scenario s
- x^f_{is} : final stock for plant i over scenario s
- x_{ikts} : fuel stock of nuclear power plant i during time step $t \in T$ of cycle k over scenario s
- p_{ikts} : production of nuclear plant i during time step t of cycle k over scenario s
- z_{ikts} (binary) : $= 1$ iff $x_{ikts} \leq BO_{ik}$ (used for modeling `[CT6]` and `[CT12]`)
- ξ_{ikts} (binary) : $= 1$ iff $PB_{ik}(x_{ikts})\underline{P}_{it}D^t \leq x_{ikts}$ (used for `[CT6]`)

As in [59] we introduce intermediate variables $\eta(i,w,k) = \begin{cases} 1 - \sum_{w' \leq w} y_{iw'2} & k = 1 \\ \sum_{w' \leq w - L_{ik}} y_{iw'k} & k = |K_i| \\ \sum_{w' \leq w - L_{ik}} y_{iw'k} - \sum_{w' \leq w} y_{iw'(k+1)} & \text{otherwise} \end{cases}$

This binary variable indicates whether week w is a production week in cycle k for plant i.

Note that ha_{ik} can be rewritten as $ha_{ik} = \sum_{w \in W} w y_{iwk}$.

$$z = \min \sum_{i \in I} \sum_{k \in K} C_{ik} r_{ik} + \frac{1}{|S|} \sum_{s \in S} \left(\sum_{t \in T} \sum_{j \in J} C_{jts} p_{jts} - \sum_{i \in I} C^f_i x^f_{is} \right) \tag{A.1}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{k \in K_i(w(t))} p_{ikts} + \sum_{j \in J} p_{jts} = DEM_{ts}, \qquad \forall t \in T, s \in S \tag{A.2}$$

$$\underline{P}_{jts} \leq p_{jts} \leq \overline{P}_{jts} \qquad \forall j \in J, t \in T, s \in S \tag{A.3}$$

$$\sum_{(iwk) \in G} y_{iwk} \leq \alpha_G \qquad \forall G \in \mathcal{G} \tag{A.4}$$

$$T^o_{ik} \leq \sum_{w \in W} w y_{iwk} \leq T^a_{ik} \qquad \forall i \in I, k \in K_i \tag{A.5}$$

$$p_{ikts} \leq \overline{P}_{it}\eta(i, w(t), k), \qquad \forall i \in I, k \in K_i, t \in T, s \in S \tag{A.6}$$

$$Y_{ik} = \sum_{w \in W} y_{iwk} \qquad \forall i \in I, k \in K_i \tag{A.7}$$

$$\underline{R}_{ik} Y_{ik} \leq r_{ik} \leq \overline{R}_{ik} Y_{ik} \qquad \forall i \in I, k \in K_i \tag{A.8}$$

$$Y_{i(k-1)} \geq Y_{ik} \qquad \forall i \in I, \forall k \in K_i \setminus \{1\} \tag{A.9}$$

$$x^1_{i(k+1)s} = x^2_{iks} - \sum_{t \in T} p_{ikts} D^t \qquad \forall i \in I, k \in K_i, s \in S \tag{A.10}$$

$$x^2_{iks} = \left[q_{ik}(x^1_{iks} - \texttt{BO}_{i(k-1)}) + r_{ik} + \texttt{BO}_{ik}\right] Y_{ik} + x^1_{iks}(1 - Y_{ik}) \qquad \forall i \in I, k \in K_i, s \in S \tag{A.11}$$

$$x^1_{iks} \leq A_{ik} + (1 - Y_{ik})N \qquad \forall i \in I, k \in K_i, s \in S \tag{A.12}$$

$$x^2_{iks} \leq S_{ik} + (1 - Y_{ik})N \qquad \forall i \in I, k \in K_i, s \in S \tag{A.13}$$

$$x^f_{is} = \sum_{k \in K_i} x_{i(k+1)s}(Y_{ik} - Y_{i(k+1)}) \qquad \forall i \in I, s \in S \tag{A.14}$$

$$x_{ikts} = x^2_{iks} - \sum_{t' \leq t} p_{ikt's} D^t \qquad \forall i \in I, k \in K_i, t \in T, s \in S \tag{A.15}$$

$$\sum_{t \in T} (\underline{P}_{it} - p_{ikts}) D^t (1 - z_{ikts}) \leq \overline{M}_{ik} \qquad \forall i \in I, k \in K_i \tag{A.16}$$

$$(1 - \epsilon)\xi_{ikts} z_{itks} \leq \frac{p_{ikts}}{PB_{ik}(x_{ikts})\overline{P}_{it}} \xi_{ikts} z_{itks} \leq (1 + \epsilon)\xi_{ikts} z_{itks} \qquad \forall i \in I, k \in K_i, t \in T, s \in S \tag{A.17}$$

$$\eta(i, w, k)(BO_{ik} - x_{itks}) z_{ikts} \geq 0, \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.18}$$

$$\eta(i, w, k)(x_{itks} - BO_{ik})(1 - z_{ikts}) \geq 0, \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.19}$$

$$(PB_{ik}(x_{itks})\overline{P}_{it} D_t - x_{itks})\xi_{itks} \leq 0 \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.20}$$

$$(x_{itks} - PB_{ik}(x_{itks})\overline{P}_{it} D_t)(1 - \xi_{itks}) \leq 0 \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.21}$$

$$y_{iwk}, Y_{ik} \in \{0,1\} \qquad \forall i \in I, \forall k \in K_i, \forall w \in W \tag{A.22}$$

$$z_{ikts}, \xi_{ikts} \in \{0,1\} \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.23}$$

$$p_{ikts}, p_{jts}, x^1_{iks}, x^2_{iks}, x^f_{is}, x_{ikts} \geq 0 \qquad \forall i \in I, \forall k \in K_i, \forall t \in T, \forall s \in S \tag{A.24}$$

The objective function (A.1) minimizes the sum of the total reload cost and the average production cost (reduced by the profit value of remaining fuel) over all scenarios. Constraints (A.2) specify that the total quantity produced by nuclear power plants and thermal power plants must be equal to the energy demand for each period and scenario. Constraints (A.4) are a general way of writing constraints `[CT14]` to `[CT21]`. The right hand side α_G is an integer number, generally equal to one (corresponding then to incompatibility constraints between dates of outages). Constraints (A.5) ensure that outages start in the imposed intervals of `[CT13]`. Constraints (A.3) and (A.6) limit the production of thermal and nuclear power plants. Equality constraints (A.7) define the binary variables Y_{ik} that indicate whether cycle k is selected or not in the schedule. Constraints (A.8) bound the reloaded amount of fuel in a cycle. Constraints (A.9) say that when a cycle k is selected, then all cycles before k are also selected in the schedule. Constraints (A.10) say that the level of fuel stock at the end of cycle k (i.e., at the beginning of cycle $k+1$) is the level of stock after refueling minus the total production of cycle k. Constraints (A.11) express that the level of fuel stock after refueling is equal to the level before refueling (i.e., at the beginning of cycle k), plus a function of the reload quantity according to constraint `[CT10]`. The two levels of stock after and before refueling are bounded in constraints (A.12) and (A.13), corresponding to constraints `[CT11]`. $N > 0$ is a very big parameter so that no bound holds if $Y_{ik} = 0$, similarly to [59]. The computation of the final level of fuel at the end of the time horizon is made in constraint (A.14). In these constraints, only one term in the right-hand-side sum will be non-zero by (A.9) and will correspond to the final stock. Constraints (A.15) define the level of stock during a production time step as the stock level after reload minus the total production of the campaign up to this step. The modulation constraints `[CT12]` are expressed in (A.16). Constraints (A.17) `[CT6]` impose the piecewise linear decreasing production profile as soon as $PB_{ik}(x_{ikts})\underline{P}_{it}D^t \leq x_{ikts} < BO_{ik}$, which implies $\xi_{ikts}z_{ikts} = 1$ via constraints (A.18-A.21), or no production otherwise. Both later constraints use variables z_{ikts} that are forced to 0 as soon as $x_{ikts} \geq BO_{ik}$ by constraints (A.18), and to 1 otherwise by constraints (A.19). Constraints (A.20) and (A.21) ensure that the

level of fuel is enough to start the BO profile. Constraints (A.22), (A.23) and (A.24) define the range of the variables.

The complexity of the model is mainly due to the combinatorial choice for binary variables y_{iwk}, z_{itks} and ξ_{itks}, and the nonlinearity of constraints (A.11), (A.14) and (A.16) - (A.21).

A.4 Decomposition and solution method

This Section presents how the proposed approach can be derived and explained on the basis of the compact formulation and the outline of the complete solution method.

A.4.1 Decomposition

The set of constraints can be decomposed in two sets. The *global* constraints are demand constraints (A.2) and incompatibility constraints (A.4) which link all power plants, and constraints (A.3) that bound the production in thermal plants. The *local* constraints (A.5)-(A.21) can be decomposed by each nuclear power plant i.

Assume that we are able to find for each nuclear power plant i the set of all possible production plans (i.e., feasible values of $p_{ikts}, \forall t \in T$, outages and reload quantities). Define $\mathscr{P}$ as the index set of all feasible production plans for the set of nuclear power plants, and let $\mathscr{P}_i$ be the subset of production plans associated to nuclear power plant i. Therefore, we can write a set partitioning-like reformulation (i.e., a set partitioning model with some more constraints) of the original problem where binary variables y_ℓ indicates whether production plan $\ell \in \mathscr{P}_i$ is selected for plant i, and variables p_{jts} represent the thermal energy used for each scenario $s \in S$ and each period of time $t \in T$.

$$(F) \quad z(F) = \min \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} c_\ell y_\ell + \frac{1}{|S|} \sum_{s \in S} \sum_{t \in T} \sum_{j \in J} C_{jts} p_{jts} \tag{A.25}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} p_{\ell ts} y_\ell + \sum_{j \in J} p_{jts} = DEM_{ts}, \qquad \forall t \in T, \forall s \in S \tag{A.26}$$

$$\sum_{\ell \in \mathscr{P}_i} y_\ell = 1 \qquad \forall i \in I \tag{A.27}$$

$$\underline{P}_{jts} \leq p_{jts} \leq \overline{P}_{jts}, \qquad \forall j \in J, \forall t \in T, \forall s \in S \tag{A.28}$$

$$\sum_{\ell \in \mathscr{P}} a_{\ell G} y_\ell \leq \alpha_G, \qquad \forall G \in \mathcal{G} \tag{A.29}$$

$$y_\ell \in \{0,1\} \qquad \forall \ell \in \mathscr{P}, \tag{A.30}$$

$$p_{jts} \geq 0 \qquad \forall j \in J, \forall t \in T, \forall s \in S \tag{A.31}$$

where $p_{\ell ts}$ is the quantity produced in plan ℓ in time step t over scenario s, and $a_{\ell G}$ are 0-1 coefficients associated with plan ℓ in constraint set G.

Equation (A.25) is the resulting objective function to be minimized. Constraints (A.26) specify that the energy demand of each period $t \in T$ and for each scenario $s \in S$ must be satisfied. Constraints (A.27) impose that for each nuclear power plant one and only one column (refuel and production plan) must be chosen. Constraints (A.28) bound the quantity of energy that a thermal power plant can produce each time period. Constraints (A.29) limit the number of nuclear power plants that can stop together. Constraints (A.30) and (A.31) define the range of the variables.

Note that constraints (A.26), (A.28) and (A.29) correspond to constraints (A.2), (A.3) and (A.4).

A.4.2 Solution method

The efficiency of the approach largely depends on the tractability of problem (F) and the $|I|$ subproblems associated with each nuclear power plant. These problems have very large scale given the number $|S|$ of scenarios and the number T of time steps. To deal with this complexity, we chose to simplify the original set of data by aggregating the S scenarios into a single *average* scenario $\bar{s}$, and the time steps $t = 1, \ldots, T$ into weeks $w = 1, \ldots, W$. We note for every week $w \in W$:

$$\begin{aligned}
DEM_w &= \frac{1}{|S|} \sum_{s \in S} \sum_{t \in T_w} D^t . \, DEM_{ts} \\
C_{jw} &= \frac{1}{|S|} \sum_{s \in S} \frac{|W|}{|T|} \sum_{t \in T_w} C_{jts} \\
\underline{E}_{jw} &= \frac{1}{|S|} \sum_{s \in S} \sum_{t \in T_w} D^t . \, \underline{P}_{jts} \\
\overline{E}_{jw} &= \frac{1}{|S|} \sum_{s \in S} \sum_{t \in T_w} D^t . \, \overline{P}_{jts}
\end{aligned}$$

$$\overline{E}_{iw} \;=\; \sum_{t \in T_w} \overline{P}_{it} \,.\, D^t$$

From now on, production quantities stand for each week $w \in W$, and we use notations $e_{\ell w}$ for the production of column $\ell \in \mathscr{P}_i$ in week w, and production variables e_{jw} (resp. e_{ikw}) for thermal (resp. nuclear) power plants. Recall that the energy is related to the power by the formula : $E_w = \sum_{t \in T_w} p_t D^t$. Therefore the problem (F) to solve becomes

$$(\bar{F}) \quad z(\bar{F}) = \min \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} c_\ell y_\ell + \sum_{w \in W} C_{jw} e_{jw} \tag{A.32}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} e_{w\ell} y_\ell + \sum_{j \in J} e_{wj} = DEM_w \qquad \forall w \in W \tag{A.33}$$

$$\sum_{\ell \in \mathscr{P}_i} y_\ell = 1 \qquad \forall i \in I \tag{A.34}$$

$$\underline{E}_{jw} \leq e_{jw} \leq \overline{E}_{jw} \qquad \forall j \in J, \forall w \in W \tag{A.35}$$

$$\sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} a_{\ell G} y_\ell \leq \alpha_G \qquad \forall G \in \mathcal{G} \tag{A.36}$$

$$y_\ell \in \{0,1\} \qquad \forall \ell \in \mathscr{P} \tag{A.37}$$

$$e_{jw} \geq 0 \qquad \forall j \in J, w \in W \tag{A.38}$$

Formulation $(\bar{F})$ is impractical to solve even for instances of moderate size, since the number of variables is typically exponential, which justifies a column generation approach. We define a linear relaxation $(L\bar{F})$ of (F):

$$(L\bar{F}) \quad z(L\bar{F}) = \min \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} c_\ell y_\ell + \sum_{w \in W} C_{jw} e_{jw} \tag{A.39}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{\ell \in \mathscr{P}_i} e_{\ell w} y_\ell + \sum_{j \in J} e_{jw} = DEM_w \qquad \forall w \in W \tag{A.40}$$

$$\sum_{\ell \in \mathscr{P}_i} y_\ell = 1 \qquad \forall i \in I \tag{A.41}$$

$$\underline{E}_{jw} \leq e_{jw} \leq \overline{E}_{jw} \qquad \forall j \in J, w \in W \tag{A.42}$$

$$0 \leq y_\ell \leq 1 \qquad \forall \ell \in \mathscr{P}, \tag{A.43}$$

$$e_{jw} \geq 0 \qquad \forall j \in J, \forall w \in W \tag{A.44}$$

When passing from (F) to $(L\bar{F})$, constraints (A.36) have been removed since their dual variables are hard to compute in the pricing problem within the limited computational time

imposed by the competition. For compensation we allow to add to problem $(L\bar{F})$ some columns with positive reduced costs as described in Section A.7. The solution of problem $(L\bar{F})$ provides the dual variables μ_w and u_i associated with constraints (A.33) and (A.34), respectively. The pricing problem exploits them for generating the column(s) with minimum reduced cost:

$$\ell^* = \underset{\ell \in \mathscr{P}}{\operatorname{argmin}}\, \bar{c}_\ell \quad \text{with} \quad \bar{c}_\ell = c_\ell - \sum_{w \in W} e_{\ell w}\mu_w - u_i \tag{A.45}$$

The pricing subproblem (P_i) associated with each nuclear power plant $i \in I$ can be written as follows, given the decomposition previously described and aggregation of time steps into weeks. Indexes of various scenarios have disappeared and the t indexes have been replaced by w indexes in data and variables.

$$(P_i) \quad \min \sum_{k \in K_i} C_{ik} r_{ik} - C_i^f x_i^f - \sum_{w \in W, k \in K_i(w)} e_{ikw}\mu_w - u_i \tag{A.46}$$

$$\text{s.t.} \quad e_{ikw} \le \overline{E}_{iw}\eta(i,w,k), \qquad k \in K_i, w \in W \tag{A.47}$$

$$Y_{ik} = \sum_{w \in W} y_{ikw} \qquad \forall k \in K_i \tag{A.48}$$

$$\underline{R}_{ik} Y_{ik} \le r_{ik} \le \overline{R}_{ik} Y_{ik} \qquad \forall k \in K_i \tag{A.49}$$

$$Y_{i(k-1)} \ge Y_{ik} \qquad \forall k \in K_i \setminus \{1\} \tag{A.50}$$

$$x^1_{i(k+1)} = x^2_{ik} - \sum_{w \in W} e_{ikw} \qquad \forall k \in K_i \tag{A.51}$$

$$x^2_{ik} = \left[q_{ik}(x^1_{ik} - \mathtt{BO}_{i(k-1)}) + r_{ik} + \mathtt{BO}_{ik}\right] Y_{ik} + x^1_{ik}(1 - Y_{ik}) \qquad \forall k \in K_i \tag{A.52}$$

$$x^1_{ik} \le A_{ik} + (1 - Y_{ik})N \qquad \forall k \in K_i \tag{A.53}$$

$$x^2_{ik} \le S_{ik} + (1 - Y_{ik})N \qquad \forall k \in K_i \tag{A.54}$$

$$x_i^f = \sum_{k \in K_i} x^1_{i(k+1)}(Y_{ik} - Y_{i(k+1)}) \tag{A.55}$$

$$\sum_{w \in W'_{ik}} (\overline{E}_{iw} - e_{ikw}) z_{ikw} \le \overline{M}_{ik} \qquad \forall k \in K_i \tag{A.56}$$

$$(1-\epsilon)\xi_{ikt} z_{itk} \le \frac{p_{ikt}}{PB_{ik}(x_{ikt})\overline{P}_{it}} \xi_{ikt} z_{itk} \le (1+\epsilon)\xi_{ikt} z_{itk} \qquad \forall i \in I, k \in K_i, t \in T \tag{A.57}$$

$$\eta(i,w,k)(BO_{ik} - x_{ikt}) z_{ikt} \ge 0, \qquad \forall i \in I, \forall k \in K_i, \forall t \in T \tag{A.58}$$

$$\eta(i,w,k)(x_{ikt} - BO_{ik})(1 - z_{ikt}) \ge 0, \qquad \forall i \in I, \forall k \in K_i, \forall t \in T \tag{A.59}$$

$$(PB_{ik}(x_{ikt})\overline{P}_{it} D_t - x_{ikt})\xi_{itk} \le 0 \qquad \forall i \in I, \forall k \in K_i, \forall t \in T \tag{A.60}$$

$$(x_{ikt} - PB_{ik}(x_{ikt})\overline{P}_{it} D_t)(1 - \xi_{itk}) \le 0 \qquad \forall i \in I, \forall k \in K_i, \forall t \in T \tag{A.61}$$

$$x_{ikt} = x^2_{ik} - \sum_{\tau < t} p_{ik\tau} D^\tau \qquad \forall k \in K_i, \forall t \in T \tag{A.62}$$

$$\sum_{t \in T_w} p_{ikt} D_t = e_{ikw} \sum_{w' \le w} (y_{ikw'} - y_{i(k+1)w'}) \qquad \forall k \in K_i, \forall w \in W \tag{A.63}$$

$$y_{iwk}, Y_{ik}, z_{ikt}, \xi_{ikt} \in \{0,1\} \qquad \forall k \in K_i, \forall w \in W, \forall t \in T \qquad \text{(A.64)}$$

$$e_{ikw}, p_{ikt}, x^1_{ik}, x^2_{ik}, x^f_i, x_{ikt} \geq 0 \qquad \forall k \in K_i, \forall w \in W, \forall t \in T \qquad \text{(A.65)}$$

The Pricing problem (P) consists in optimizing every problem (P_i) for $i \in I$ and output the column with best optimal value (minimum reduced cost). The columns with negative reduced cost output by (P) are added to problem $(L\bar{F})$ and the process iterates between $(L\bar{F})$ and (P) until (P) finds no more negative reduced cost column. As we mentioned in Section A.1, the proposed column generation method is a heuristic scheme. Indeed, due to the time constraints imposed by the competition, only an approximation of the pricing problem P is solved at each iteration. This holds because the pricing subproblem is a shortest path problem solved on a graph whose paths do not explicitly represent all feasible plans, as explained in Section A.5. At the end of the column generation, the set of columns forms a relaxed restricted master program which is called the Restricted $L\bar{F}$ $(RL\bar{F})$, and solved to optimality as explained in Section A.6.

The obtained optimal solution is a set of columns and variables p_{wj} satisfying constraints (A.40)-(A.42) at minimum cost. Each column is a production plan containing the dates for the outages and the periods when stock reaches a critical level equal to BO_{ik} (if such periods exist for a given nuclear plant). Therefore the solution obtained at the end of the first phase is used for fixing the decision variables on the outages and the date when the BO-threshold is attained. The advantage is to avoid to deal explicitly with non linear constraints, since once one has fixed BO, the corresponding decreasing profile is completely defined for the successive periods until the next outage.

The only decisions that remain to be made in the second phase are the reload variables r_{ik} and the production variables p_{ikts} , which are easier to settle once the binary outage variables y_{iwk} have been fixed. For setting the values of p_{iktsi}, p_{jts} and r_{ik}, it is necessary to use the whole model with all time steps $t = 1, \ldots, T$ and all scenarios $s \in S$. Since it has no more binary variables, it can be solved efficiently by Linear Programming.

The two-stage method is stated below.

Phase 1 Fixing dates for outages and BO stock-level periods

subphase 1 Solve the aggregate set partitioning-like formulation $(L\bar{F})$ by column generation

- Aggregate data, passing from $|T \times S| = (5000 \times 500)$ to $|W \times \{\bar{s}\}| = (260 \times 1)$
- Create an acyclic graph G_i, for each nuclear power plant i

- Repeat until a time limit is elapsed or no new column of negative reduced cost exists
  - Solve a shortest path subproblem in G_i, for each nuclear power plant i
  - Add the new columns (corresponding to shortest paths) to the $(L\bar{F})$ master problem and solve it by the Simplex method

subphase 2 Give the final restricted master problem to a MIP solver adding all the clique-like inequalities (A.36), exploiting the fact that only a small number of nuclear power plants can be refueled at the same time. Outputs are the dates for outages and `BO` stock-level periods.

Phase 2 Solve LP problems for defining the reload level for each date of outages and optimizing the production planning for each power plant and each scenario.

Phase 2 is described in Section A.8. Sections A.5 and A.6 detail the first phase of the method. In phase 1, for every nuclear plant $i \in I$, two decisions are fixed: dates for outages respecting the scheduling constraints on outages and periods when stock reaches the BO_{ik} level, with imposed non linear decreasing power profile on the following periods. This step is the most time-consuming step of the method.

A.5 Solving the pricing problem

The pricing problem (P_i) consists in finding a reload and production plan for each nuclear power plant i. The problem contains a lot sizing problem and therefore can be solved by a shortest path algorithm in a graph (see for example [85], [86] and [87]). The main point is how to build the graph. The difficulty of this step is that the decision variables are a combination of continuous and binary variables. The binary variables define the possible outage dates while the continuous variables define the quantity of fuel that must be reloaded and the production for each time step. The further difficulty is due to the `BO` piecewise non linear function. Last but not least comes from the limited computational time imposed by the competition. Therefore, in the proposed approach, we chose to design a graph in which a production plan would be made of a relatively small number of arcs, each production campaign being represented by at most two arcs, and in such a way that cost computations of each arc could be managed efficiently. For efficiency reasons, we made the simplification of discretizing the possible fuel stock levels at the beginning of outages, so that the stock level could be fixed once for all at both ends of

each arc of the graph. In the construction of the graph constraints (A.47) to (A.61) are taken into account either when a node is defined or either when an arc is defined, such that any path on this graph represents a feasible solution.

The pricing problem has been implemented as a three-step process:

Algorithm 4 The pricing algorithm

for all nuclear power plants i **do**

1) create an acyclic static *stock graph* $G_i = (N_i, A_i)$;

2) for each arc $a \in A_i$ compute weight $\bar{c}_a$ (optimal reduced cost on the arc);

3) generate a new column by finding in G_i a path of minimum weight.

end for

Step 1 is quite time consuming but occurs only once at the beginning of the whole process. Steps 2 and 3 are light processes and can be serialized with the computation of new dual variables μ_w.

A.5.1 Creation of a static graph

Description of the graph. Each node $n \in N_i$ in graph G_i represents a possible state of power plant i and is characterized by a 4-tuple $(type_n, x_n, w_n, k_n)$, where:

- w_n is the week associated with n ;
- x_n is the fuel stock of power plant i *at the beginning* of week w_n ;
- k_n is the id of the campaign in which the power plant will be during week w_n ;
- $type_n$ is the node type. There are two main types of nodes:
  - an "`OUTAGE`" node n marks the start of a new production campaign and w_n is the week during which the outage occurs ;
  - a "`BO`" node n marks the start of a time period during which the fuel stock will be lower than BO_{ik_n} and the power level will be imposed by constraint `[CT6]` ; the stock x_n of a `BO` node is indeed a little higher than BO_{ik_n}, but ensures that BO_{ik_n} can be reached *during* week w_n by imposing the maximal power production of constraint `[CT5]` at the beginning of the week. In the model, it corresponds to setting the constraints (A.57)-(A.63).

  Besides, two nodes with special types are added, corresponding to the `initial` and `final` states of a power plant.

Each arc $a = (n_1, n_2) \in A_i$ in graph G_i represents a possible direct transition from state n_1 to n_2.

- If n_1 is an `OUTAGE` node, this means that it is possible to reach state n_2 by proceeding to an outage on week w_{n_1} followed by a production campaign, without need for any other outage and without reaching the `BO` stock level during the time interval $[w_{n_1}, w_{n_2}]$. Note that n_2 can be either a `BO` node or an `OUTAGE` node. If it is a `BO` node, then $k_{n_2} = k_{n_1}$; else $k_{n_2} = k_{n_1} + 1$. Arcs leaving from `OUTAGE` nodes will be referred to as `OUTAGE` arcs.
- If n_1 is a `BO` node, this means there exists a production plan leading from n_1 to n_2 that respects the imposed profile of constraint `[CT6]` with the acceptable ϵ tolerance. Note that here n_2 has to be an `OUTAGE` node and thus $k_{n_2} = k_{n_1} + 1$. Arcs leaving from `BO` nodes will be referred to as `BO` arcs.

Furthermore, arcs are constructed so as to guarantee that technical constraints `[CT3]` to `[CT13bis]` (eq. (A.47) to (A.56)) can be respected for any path from the `initial` to the `final` node of G_i.

Building the graph. We build graph G_i in the following way:

Algorithm 5 Creation of graph G_i

1: create the `initial` node at week 1
2: **for** week $w = 1$ **to** $|W|$ **do**
3: **for all** previously created nodes n with $w_n = w$ **do**
4: create new nodes n' with $w_{n'} > w$ so that there exists a production plan leading the power plant from state n to state n' without violating constraints `[CT3]` to `[CT13bis]`
5: **end for**
6: **end for**

In order to find new nodes (step 4 of Algorithm 5), we follow the beam of possible stock interval(s) over time from a node n, as stated in Algorithm 6:

Algorithm 6 Creation of new nodes in G_i from $n = (type_n, x_n, w_n, k_n)$

1: **for** week $w = w_n + 1$ **to** $|W|$ **do**
2: compute the possible stock interval $[x_{min_{n,w}}, x_{max_{n,w}}]$ of power plant i at the beginning of week w
3: **if** the outage of campaign $k_n + 1$ is allowed at week w according to constraint `CT13` **then**
4: compute the intersection of $[x_{min_{n,w}}, x_{max_{n,w}}]$ with $[0, A_{ik_n+1}]$
5: create new nodes $n' = (\texttt{OUTAGE}, x_{n'}, w, k_n+1)$ with $x_{n'}$ in this intersection (constraint `CT11`)
6: **end if**
7: **if** $\texttt{BO}_{ik_n} \in [x_{min_{n,w}}, x_{max_{n,w}}]$ **then**
8: create new nodes $n' = (\texttt{BO}, x_{n'}, w-1, k_n)$ with $x_{n'} \in [x_{min_{n,w-1}}, x_{max_{n,w-1}}]$ computed so that if the power plant follows the imposed profile of `[CT6]` during week $w-1$, its stock will fall under $\texttt{BO}_{ik_n}$ at the beginning of week w
9: **end if**
10: **end for**

For steps 5 and 8 of Algorithm 6, we discretize the interval $[x_{\min_{n,w}}, x_{\max_{n,w}}]$ of possible stocks of fuel at week w by a fixed amount of fuel Δ_i. New nodes are only created when needed, so as to ensure that there will be a node in every non-empty intervals $[x_{\min_{n,w}}, x_{\max_{n,w}}] \cap [x, x + \Delta_i]$, for all $x \in \mathbb{R}_+$. We compute Δ_i relatively to the mean capacity of powerplant i after choosing *a priori* a global discretisation parameter ρ :

$$\Delta_i = \rho \,.\, \frac{\sum_{w \in W} \overline{E}_{iw}}{|W|}$$

Step 2 of Algorithm 6 depends on the node type.

- If $type_n = \texttt{BO}$, we follow the imposed profile of constraint `[CT6`, i.e. (A.57)`]` with the authorized ϵ-deviation, counted positively for $x_{min_{n,w}}$ and negatively for $x_{max_{n,w}}$.
- If $type_n = \texttt{OUTAGE}$, we use `[CT10`, i.e. (A.52)`]` to compute stock variation during the outage, `[CT7`,(A.49)`]` (bounds on refueling) and `[CT11`,(A.53)-(A.54)`]` (maximum stock after refueling) to compute the interval after refueling ; when the production campaign starts, $x_{max_{n,w}}$ is computed using the maximum allowed modulation of `[CT12`,(A.56)`]` – and $x_{min_{n,w}}$ using no modulation at all.

The node creation process is illustrated by figure A.1 for the second case (starting from an `OUTAGE` node).

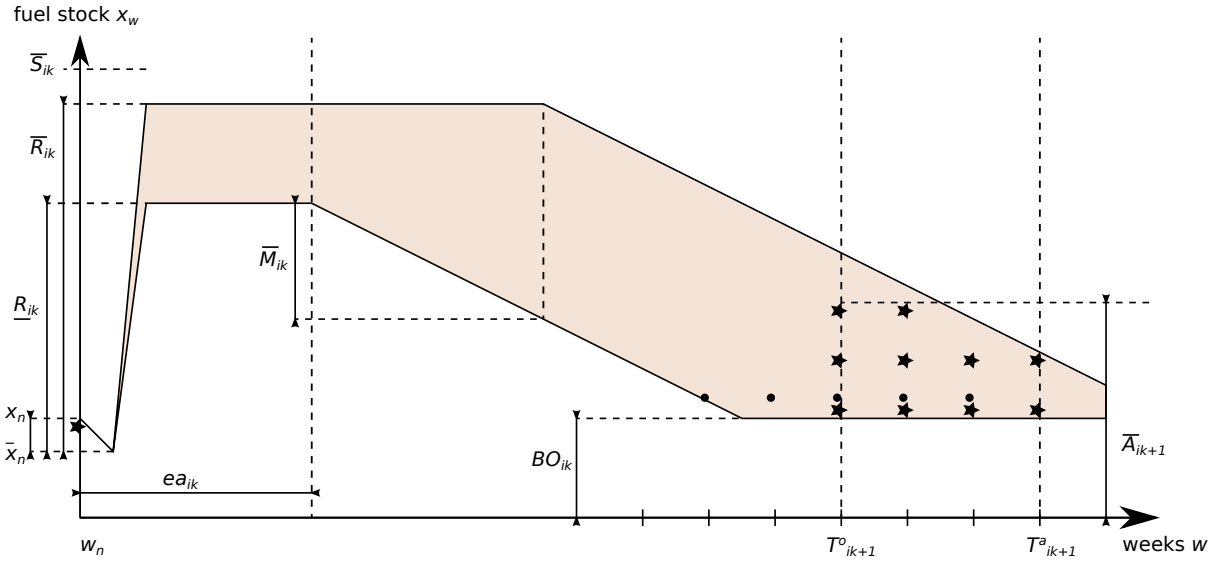

Figure A.1: Creating new nodes from an `OUTAGE` node n of campaign k in graph G_i inside the beam of possible stock intervals. Stars inside the beam correspond to the creation of new `OUTAGE` nodes associated with campaign $k+1$, whereas circles are new `BO` nodes associated with campaign k. Note that no `BO` node is created at week $T^a_{i(k+1)}$, because the paths that would go through such a node would violate constraint `CT13bis`.

When all nodes have been created by the process of Algorithm 5, a second pass takes place in order to create the arcs of the graph between all nodes that can be connected.

A.5.2 Weighting the arcs of a graph

The goal of this graph is to compute a production plan $\ell \in \mathscr{P}_i$ for power plant i with minimal reduced cost $\bar{c}_\ell$, which amounts to find according to equation (A.45) :

$$\min_{r,e,x_i^f} \sum_{k \in K} C_{ik} r_{ik} - C_i^f x_i^f - \sum_{w \in W} \mu_w e_w - u_i$$

where

- x_i^f is the quantity of remaining fuel at the end of the timeline in the aggregated scenario ;
- e_w is the energy produced at week w in production plan ℓ ;
- μ_w is the dual variable associated to constraint (A.33) ;
- u_i is the dual variable associated to constraint (A.34)

Let us recall that a production plan is supported by some unique path $\mathcal{A}_\ell \subset A_i$ in graph G_i, and that each campaign k in a production plan is carried either by a single `OUTAGE` arc, or by

an OUTAGE arc followed by a BO arc.

It comes that the previous expression can be rewritten as:

$$\min \bar{c}_\ell = \sum_{a=(n,n')\in\mathcal{A}_\ell} \left(C_{ik_n} r_a - C_i^f x_a^f - \sum_{w\in W(a)} \mu_w e_w \right) - u_i$$

where:

– r_a is the refuel quantity at the beginning of arc a, i.e. :

$$r_a = \begin{cases} r_{ik_n} & \text{if } a \text{ is an } \texttt{OUTAGE} \text{ arc} \\ 0 & \text{if } a \text{ is a } \texttt{BO} \text{ arc} \end{cases}$$

– x_a^f is the remaining fuel at the end of the timeline if a is the final arc in path $\mathcal{A}_\ell$, i.e. :

$$x_a^f = \begin{cases} x_i^f & \text{if } type_{n'} = \texttt{FINAL} \\ 0 & \text{otherwise} \end{cases}$$

– $W(a) = [w_n, w_{n'}[$ is the set of weeks spanned by arc a.

With this rewriting, we see that each variable that plays a part in the reduced cost of path $\mathcal{A}_\ell$ is *uniquely linked* with a variable associated with some arc of ℓ.

Let us call the *reduced cost* of arc $a \in A_i$ the value c_a defined by:

$$\bar{c}_a = \min_{r,e,x_a^f} C_{ik_n} r_a - C_i^f x_a^f - \sum_{w\in W(a)} \mu_w e_w \tag{A.66}$$

While ensuring that constraints of (P) hold true for every path in ℓ, we can easily find the production plan with minimum reduced cost by solving a shortest path in the graph. The reduced cost of this production plan is obtained by substracting u_i from the minimum weight path value.

Determining the weight of a BO arc is straightforward because there is no refuel at the beginning of such an arc ($r_a = 0$) and the productions $e_{(W(a))}$ are fixed for each timestep by constraint [CT6, i.e. (A.57)], if we disregard the allowed deviation ϵ. If the arc is FINAL, we can compute x_a^f by substracting the productions from its initial fuel stock. The weight of a BO arc a is given by the following expressions, where everything is known:

$$\bar{c}_a = \begin{cases} -C_i^f \,.\, \left(x_n - \sum_{w\in W(a)} e_w \right) - \sum_{w\in W(a)} \mu_w \,.\, e_w & \text{if } a \text{ is } \texttt{FINAL} \\ -\sum_{w\in W(a)} \mu_w \,.\, e_w & \text{otherwise} \end{cases} \tag{A.67}$$

Determining the weight of a non final `OUTAGE` arc is more intricate because the allowed modulation gives some latitude to productions $e_{(W(a))}$. However, the campaign k_n, the production weeks $W(a) = [w_n, w_{n'}[$ and the levels of fuel $(x_n, x_{n'})$ at the beginning and at the end of the arc are known.

Following equation (A.52), let $\bar{x}_n = \left(\frac{Q_{ik_n}-1}{Q_{ik_n}}(x_n - \texttt{BO}_{ik_n}) + \texttt{BO}_{ik_n-1}\right)$. The problem can then be stated as follows:

$$(LP_a) \quad \bar{c}_a = \min_{r_a, e_{(W(a))}} C_{ik_n} r_a - \sum_{w \in W(a)} \mu_w e_w \tag{A.68}$$

$$\text{s.t.} \quad \bar{x}_n + r_a = x_{n'} + \sum_{w \in W(a)} e_w \qquad [\texttt{from(A.51)}] \tag{A.69}$$

$$0 \leq e_w \leq \overline{E}_{iw} \qquad \forall w \in W(a) \quad [\texttt{from(A.47)}] \tag{A.70}$$

$$\underline{R}_{ik_n} \leq r_a \leq \overline{R}_{ik_n} \qquad [\texttt{from(A.49)}] \tag{A.71}$$

$$\bar{x}_n + r_a \leq \overline{S}_{ik_n} \qquad [\texttt{from(A.54)}] \tag{A.72}$$

$$\sum_{w \in W(a)} (\overline{E}_{iw} - e_w) \leq \overline{M}_{ik_n} \qquad [\texttt{from(A.56)}] \tag{A.73}$$

We can use (A.69) to rewrite r_a and remove it from the problem. It comes that we are looking for the minimum under constraints (A.70) to (A.73) of:

$$\bar{c}_a = C_{ik_n}\left(x_{n'} - \bar{x}_n\right) + \sum_{w \in W(a)} \left(C_{ik_n} - \mu_w\right) e_w$$

As the first part of the sum is *known* for each arc, we only have to find the productions e_w that minimize $\sum_{w \in W(a)} \left(C_{ik_n} - \mu_w\right) e_w$. We tried two methods to achieve this goal:

The exact solution is obviously given by the Simplex method, since (LP_a) is a linear program, or by Algorithm 7 below, where *nullify* has to be understood as "set as close to zero as possible without violating constraints (A.70) to (A.73)":

Algorithm 7 Exact algorithm for minimizing arc weight

set all e_w to $\overline{E}_{iw}$

order the set of weeks by decreasing values of $(C_{ik_n} - \mu_{w_m}) : w_1, w_2 \ldots$

start with $m = 1$

while constraints (A.70) to (A.73) are not satisfied **do**

nullify e_{w_m}

increment m

end while

while $(C_{ik_n} - \mu_{w_m}) > 0$ **and** constraints (A.70) to (A.73) can be satisfied **do**

nullify e_{w_m}

increment m

end while

However, we encountered two problems with the exact method:

- the overall column generation process was converging too fast, and no new column was generated after about two or three iterations ; this can be overcome however by the *diversification* strategy exposed in Section A.7 ;
- this method was 3 to 10 times slower than the heuristic exposed below (depending on the instances), because it has to walk through the ordered set of weeks for each arc. Thus, on some instances, we were not able to produce enough columns in the imposed time and ended with no valid solution when using this exact algorithm.

We used a heuristic to obtain a good solution, even if not optimal: we chose to *spread* the modulation on all weeks proportionally to $\overline{E}_{iw}$, but with a different factor whether the *reduced unit cost* $\bar{c}_{ikw} = C_{ik} - \mu_w$ is positive or negative. For that purpose, we split the set of weeks into two sets: $W_k^+ = \{w \in W \text{ s.t. } C_{ik} - \mu_w > 0\}$ and $W_k^- = \{w \in W \text{ s.t. } C_{ik} - \mu_w \leq 0\}$. Then, when looking for the minimal reduced cost of our arc a, we consider two variables α_a and β_a in the $[0, 1]$ interval and we *set* the values of e_w from these variables:

$$\forall w \in W(a) \cap W_{k_n}^+, \quad e_w = \alpha_a \,.\, \overline{E}_{iw}$$
$$\forall w \in W(a) \cap W_{k_n}^-, \quad e_w = \beta_a \,.\, \overline{E}_{iw}$$

The problem becomes: find the factors α_a and β_a that minimize the expression:

$$\alpha_a \left(\sum_{w \in W(a) \cap W^+_{k_n}} (C_{ik_n} - \mu_w) \overline{E}_{iw} \right) + \beta_a \left(\sum_{w \in W(a) \cap W^-_{k_n}} (C_{ik_n} - \mu_w) \overline{E}_{iw} \right)$$

The parenthesized expressions do not have to be computed for each arc. Instead, we can compute them *in advance* for each $(k, [w, w'[)_{w<w'}$ couple. Similarly, the constraints (A.70) to (A.73) can be rewritten as linear constraints on α_a and β_a and we save a lot of computing overhead by precomputing $\sum_{w \in [w,w'[} \cap W^-_k \overline{E}_{iw}$ for each couple $(k, [w, w'[)_{w<w'}$ and their counterparts on W^+_k.

As it reduces the number of variables to a couple, this heuristic has proven to be more efficient and effective than looking for the exact solution, when placed in the context of this column generation problem : even if the minimum reduced cost is not reached by this computation, the induced time savings allow the production of a greater set of columns amongst which better combinations can be found.

A.5.3 Obtaining dual variables μ_w and u_i

At each iteration we solved $RL\bar{F}$ to optimality and the dual variables μ_w and u_i are respectively associated with constraints (A.33) and (A.34). For the first run the variables are arbitrarily chosen as follows: $u_i = 0$ for all $i \in I$ and $\mu_w = \max_{i \in I, k \in K_i} \{C_{ik} + 1\}$ so that $W^+_k = W$ for all power plants. This will create a first pool of production plans where nuclear power plants are used at their maximal capacity.

A.6 Solving the final master problem

Once $L\bar{F}$ has been solved to optimality by the Column Generation process, the following last restricted $\bar{F}$ is solved to optimality

$$(R\bar{F}) \quad z(\bar{F}) = \min \sum_{i \in I} \sum_{\ell \in \widehat{\mathscr{P}_i}} c_\ell y_\ell + \sum_{w \in W} C_{jw} e_{jw} \tag{A.74}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{\ell \in \widehat{\mathscr{P}_i}} y_\ell e_{\ell w} + \sum_{j \in J} e_{jw} = DEM_w \qquad \forall w \in W \tag{A.75}$$

$$\sum_{\ell \in \widehat{\mathscr{P}_i}} y_\ell = 1 \qquad \forall i = 1, \ldots, I \tag{A.76}$$

$$\underline{E}_{jw} \leq e_{jw} \leq \overline{E}_{jw} \qquad \forall j \in J, \forall w \in W \qquad \text{(A.77)}$$

$$\sum_{i=1}^{I} \sum_{\ell \in \widehat{\mathscr{P}}_i} a_{\ell G} y_\ell \leq \alpha_G \qquad \forall G \in \mathcal{G} \qquad \text{(A.78)}$$

$$y_\ell \in \{0,1\} \qquad \forall \ell \in \widehat{\mathscr{P}} \qquad \text{(A.79)}$$

$$e_{jw} \geq 0 \qquad \forall j \in J, \forall w \in W \qquad \text{(A.80)}$$

where $\widehat{\mathscr{P}}$ contains the subset of production plans generated from the beginning. This MIP includes constraints (A.78) which are the incompatibility constraints for some subsets $\widehat{\mathscr{P}}$ of production plans, corresponding to constraints `[CT14-21]` on the scheduling of outages. These constraints are generated dynamically during the column generation process. All of them are based on the same principle: given a set of columns, it is possible to build an incompatibility graph and we look for the maximal clique on this graph. The graph has a node for each column and an edge between any pair of incompatible columns (i.e. columns that cannot be at the same time in the optimal solution due to constraints `[CT14-21]`).

If $\ell \in \mathscr{P}_i$ denotes the column selected for each nuclear plant i in the optimal MIP solution, then the variables y_{iwk}, z_{itks} and ξ_{itks}, representing the dates of outages, the dates when the stock reaches the `BO`-level and the corresponding imposed decreasing profile for the following weeks, are fixed for the rest of the whole method.

A.7 Columns diversification

The main difficulty encountered with the method described so far comes when solving the MIP for column selection since constraints `[CT14-21]` are not part of the column generation process. We are not able to find a proper subset of columns that satisfies these constraints, most of the time. To overcome this problem, we used an alternate scheme for weighting the columns: instead of using weights $\bar{c}_a$ given by equations (A.67) and (A.68), we compute and use weights $\bar{c}'_a$ in the following way:

$$\bar{c}'_a = \begin{cases} 0 & \text{for } \texttt{BO} \text{ arcs} \\ nbConflicts(a, col) & \text{for } \texttt{OUTAGE} \text{ arcs} \end{cases} \qquad \text{(A.81)}$$

where $nbConflicts(a, cols)$ heuristically figures out the current number of (column, constraint) couples that would conflict with a new column going through arc a.

More precisely, we compute $\bar{c}'_a$ with the following algorithm:

Algorithm 8 Alternate weight computation during a *diversification* pass for an `OUTAGE` arc $a = (n, n')$

$\bar{c}'_a \leftarrow 0$

for all existing column $\ell \in \cup_i \mathscr{P}_i$ **do**

for all constraint m of type `[CT14]` to `[CT18]` **do**

if column ℓ conflicts with an outage of power plant i_a starting at week w_n by constraint m **then**

$\bar{c}'_a \leftarrow \bar{c}'_a + 1$

end if

end for

for all constraint m of type `[CT19]` **do**

if for some outage k of ℓ, (i_a, k_a) and (i_ℓ, k) are part of set G_m^{19} **and** w_n is interval $[ha_{i_\ell k} + w^m_{i_\ell k} - w^m_{i_a k_a}, ha_{i_\ell k} + w^m_{i_\ell k} - w^m_{i_a k_a} + i^m_{i_\ell k}]$ **then**

$\bar{c}'_a \leftarrow \bar{c}'_a + \frac{1}{Q_m^{19}}$

end if

end for

for all constraint m of type `[CT20]` concerning week w_m **do**

if $w_m \in [w_n, w_n + L_{i_a k_a}] \cap [ha_{i_\ell k}, ha_{i_\ell k} + L_{i_\ell k}]$ for some outage k of column ℓ **then**

$\bar{c}'_a \leftarrow \bar{c}'_a + \frac{1}{Q_m^{20}}$

end if

end for

end for

for all constraint m of type `[CT21]` concerning time period $[b_m, e_m]$ such that $i_a \in C_m^{21}$ **do**

$$\bar{c}'_a \leftarrow \bar{c}'_a + \max_{w \in [w_n, w_n + L_{i_a k_a}] \cap [b_m, e_m]} \left(\sum_{(\ell,k) \in \cup_i (\mathscr{P}_i \times K_i) \ / \ i_a \neq i_\ell \ \wedge \ i_\ell \in C_m^{21} \ \wedge \ w \in [ha_{i_\ell k}, ha_{i_\ell k} + L_{i_\ell k}] \cap [b_m, e_m]} \frac{\overline{P}_{i_\ell w}}{Q_m^{21}} \right)$$

end for

Note that $\bar{c}'_a$ only depends on the power plant i_a, the week of outage w_a and the campaign k_a ; furthermore, when new columns are generated, they only modify weights $\bar{c}'_a$ by an additive value, independently of other columns. These facts allow major factorizations in the algorithm.

This method is called each time negative reduced cost columns are found through finding the shortest path on the former graph. Then, when no more negative reduced cost columns are found with the shortest path algorithm, only the diversification method is processed and the minimum path algorithm is not ran unless zero reduced cost columns are added by diversifica-

tion. Indeed, their addition could modify the value of the dual variables though not decreasing the value of the dual problem. Note that this diversification method creates positive reduced cost columns that are added if not already present.

A.8 Calculate the reload and the production quantities

Once the dates for outages and BO-profiles have been fixed as described in Section A.6, it is still possible to improve the value of the optimal solution by looking for the optimal value of the remaining variables which are continuous. In other words, we have to determine the optimal reload quantities r_{ik} during each outage k for every nuclear plant i, and the optimal production values p_{jts} and p_{its} for non-nuclear and nuclear plants, and finally the optimal stock values p_{its} for time steps t and scenario s such that the stock level is over the `BO` threshold.

A.8.1 LP-solving for fixing reload quantities

The reload quantities r_{ik} are determined by solving a partially-disaggregated LP where every scenario s is considered instead of the average scenario $\bar{s}$, but time steps t are still aggregated into weeks w so as to accelerate solving. The optimal reload quantities r_{ik} are found by solving a linear program (LP_R) with CPLEX, where non-negative decision variables are reload variables r_{ik}, production variables e_{jws} and e_{iws}, and stock variables x_{iWs}. The constraints in this LP are all constraints `[CT1-12]`, but for weeks w instead of time steps t. Let us define the binary values $\eta^{\#}(i,w,k)$ and $Y_{ik}^{\#}$ as the values of $\eta(i,w,k)$ and Y_{ik} after they have been fixed at the end of Phase 1.

Since the two former values are known, now the outage periods have been fixed.

Let us also define the energy demand for week $w \in W$ for scenario $s \in S$ as

$$DEM_{ws} = \sum_{t \in T_w} D^t . DEM_{ts}$$

and the maximum production for power plant $i \in I$ over week $w \in W$ as

$$\overline{E}_{iw} = \sum_{t \in T_w} \overline{P}_i^t . D^t$$

and the maximum and minimum productions for power plant in $j \in J$ and scenario $s \in S$ over the week $w \in W$ as

$$\overline{E}_{jws} = \sum_{t \in T_w} \overline{P}_{jts} . D^t$$

and

$$\underline{E}_{jws} = \sum_{t \in T_w} \underline{P}_{jts} \, . \, D^t$$

and

$$C_{jws} = \frac{|W|}{|T|} \sum_{t \in T_w} C_{jts}$$

the mean cost of thermal energy production over a week $w \in W$ for $j \in J$ and $s \in S$.

The objective function is thus the following:

$$z(LP_R) = \min \sum_{i \in I} \sum_{k \in K} C_{ik} r_{ik} + \frac{1}{|S|} \sum_{s \in S} \left(\sum_{w \in W} \sum_{j \in J} C_{jws} e_{jws} - \sum_{i \in I} C_i^f x_{is}^f \right)$$

s.t.

$$\sum_{i \in I} \sum_{k \in K_i} e_{ikws} + \sum_{j \in J} e_{jws} = DEM_{ws}, \qquad \forall w \in W, \forall s \in S \tag{A.82}$$

$$e_{iws} \leq \overline{e}_{iw} \sum_{k \in K_i} \eta^{\#}(i,w,k), \qquad \forall i \in I, w \in W, \forall s \in S \tag{A.83}$$

$$\underline{E}_{jws} \leq e_{jws} \leq \overline{E}_{jws}, \qquad \forall j \in J, w \in W, s \in S \tag{A.84}$$

$$\underline{R}_{ik} Y_{ik}^{\#} \leq r_{ik} \leq \overline{R}_{ik} Y_{ik}^{\#} \qquad \forall i \in I, k \in K_i \tag{A.85}$$

$$x_{i(k+1)s}^1 = x_{iks}^2 - \sum_{w \in W'_{ik}} e_{iws} \qquad \forall i \in I, k \in K_i s \in S \tag{A.86}$$

$$x_{iks}^2 = \left[\left(\frac{Q_{ik} - 1}{Q_{ik}} \right) (x_{iks}^1 - \mathtt{BO}_{i(k-1)}) + r_{ik} + \mathtt{BO}_{ik} \right] Y_{ik}^{\#} + x_{iks}^1 (1 - Y_{ik}^{\#}) \qquad \forall i \in I, k \in K_i, s \in S \tag{A.87}$$

$$x_{iks}^1 \leq A_{ik} + (1 - Y_{ik}^{\#}) N \qquad \forall i \in I, k \in K_i, s \in S \tag{A.88}$$

$$x_{iks}^2 \leq S_{ik} + (1 - Y_{ik}^{\#}) N \qquad \forall i \in I, k \in K_i, s \in S \tag{A.89}$$

$$x_{is}^f = \sum_{k \in K_i} x_{i(k+1)s} (Y_{ik}^{\#} - Y_{i(k+1)}^{\#}) \qquad \forall i \in I, s \in S \tag{A.90}$$

$$\sum_{w \in W'_{ik}} (\overline{E}_{iw} - e_{iws}) \leq \overline{M}_{ik} \qquad \forall i \in I, k \in K_i, s \in S \tag{A.91}$$

$$e_{jws}, e_{ikws}, r_{ik}, x_{iks}^1, x_{iks}^2, x_{is}^f \geq 0 \tag{A.92}$$

This linear problem LP_R is solved for the different combinations of columns that produce the optimal value of the restricted master problem $R\bar{F}$ and the combination that gives the lowest value is kept to go to the next step where the original problem will be solved with outage periods and reload quantities fixed.

A.8.2 Disaggregate LP-solving for every scenario and initial time steps

At this final step, for every scenario $s \in S$, we go back to time steps t. Therefore, S Linear Programs (LP_s) are solved where decision variables are:

(i) the production variables p_{jts} and p_{its} for thermal and nuclear plants, respectively,

(ii) the stock variables x_{its} for time steps t such that the stock level is over the `BO` threshold (for the other time steps, the stock level is already fixed as imposed by the decreasing profile)

Constraints are the same as in the previous step, i.e. constraints `[CT1-12]`, but for time steps t now, i.e. original constraints `[CT1-12]`. The objective functions are the total production cost for each scenario s, reduced by the objective value of remaining fuel:

$$z(LP_s) = \min \sum_{t \in T} \sum_{j \in J} C_{jts} p_{jts} D^t - \sum_{i \in I} C_i^f x_{is}^f$$

The final objective value z is the refueling cost and the average total production cost reduced by the objective value of remaining fuel over all S scenarios:

$$z = \min \sum_{i \in I} \sum_{k \in K_i} C_{ik} r_{ik} + \frac{1}{|S|} \sum_{s \in S} \left(\sum_{t \in T} \sum_{j \in J} C_{jts} p_{jts} D^t - \sum_{i \in I} C_i^f x_{is}^f \right) \tag{A.93}$$

A.9 Computational results

All algorithms described in this paper were coded in C++ and compiled with the gcc (Debian 4.4.4-8) 4.4.5 compiler. CPLEX 12.1 was used as the LP solver and as the integer programming solver. The experiments were performed on a Squeeze-x64-64 (image based on Debian version sid for AMD64/EM64T) Bi AMD Opteron Dual core 3,2 Ghz with 16 GB of physical memory and 16 GB of swap memory.

A.9.1 Instances

In our computational experiments, we considered 16 instances given by EDF during the Challenge (see [72]). Almost all of them ask for a planification of 6 campaigns for a time period of 5 years. The instances have been partitioned into the following three classes.

Class 1. A set of 6 instances (A0-A5) used for the qualification phase. In these instances the number of different scenarios is at most 30 for about 20 nuclear plants and 30 other thermic power plants. The time step is the day and there are less than 27 constraints linking the nuclear power plants.

Class 2. A set of 5 instances (B6-B10) used for the second phase. In these instances the number of different scenarios considered can rise up to 121. There are about 50 nuclear plants and 20 other thermic power plants. The time step is 8 hours and there are about 140 constraints linking the nuclear power plants. There are between 114 and 235 constraints `[CT13]` on the dates of the outages.

Class 3. A set of 5 instances (X11-X15) used for the final ranking with the 5 former instances. They were not released until the end of the challenge. In these instances the number of different scenarios are 50 or 121. There are about 50 nuclear plants and 20 other thermic power plants. The time step is 8 hours and there are about 140 constraints linking the nuclear power plants. There are between 207 and 260 constraints `[CT13]` on the dates of the outages, which is the only noticeable difference with the *class 2* instances.

A.9.2 Results and discussion

In table A.1, the best results that were obtained and published on the website of the Challenge are given in the first column whereas the second column displays the ones obtained by the current method. All teams were ranked by the Challenge organizers and with the former results our method ranked 4^{th} although we were penalized by the fact that the method was not able to give answers for three of the instances. It is interesting to point out that among the first five methods proposed, our method is the only one, as far as we know, having some mathematical programming foundations.

The third column gives the best results we found since then, after the few bugs were corrected. The last column gives the parameter ρ that was chosen to obtain the former result, i.e., the parameter used to build the graphs to solve the $L\bar{F}$.

Tables A.2 and A.3 show several results about the method:

Data number	ROADEF best found results	our ROADEF results	best found results since	ρ factor
A0	8.73099e+12	8.73623e+12	8.73696e+12	0.9
A1	1.69538e+11	1.70030e+11	1.69902e+11	0.449
A2	1.46048e+11	1.46409e+11	1.46435e+11	0.9
A3	1.54430e+11	1.54711e+11	1.54747e+11	0.9
A4	1.11591e+11	1.12808e+11	1.12528e+11	0.45
A5	1.25822e+11	1.28366e+11	1.26932e+11	0.449
B6	8.34247e+10	8.47093e+10	8.46439e+10	0.5
B7	8.11742e+10	8.18391e+10	8.18501e+10	0.45
B8	8.19262e+10	8.29770e+10	8.29548e+10	0.449
B9	8.17509e+10	8.36228e+10	8.29290e+10	0.5
B10	7.77670e+10	7.87302e+10	7.88065e+10	0.449
X11	7.91168e+10	TIMEOUT	8.00418e+10	3.0
X12	7.75899e+10	7.81931e+10	7.81252e+10	0.45
X13	7.64492e+10	INFEASIBLE	7.67848e+10	0.45
X14	7.61730e+10	TIMEOUT	7.69035e+10	0.449
X15	**7.51014e+10**	**7.51014e+10**	7.49233e+10	0.449

Table A.1: Results of the ROADEF challenge

- The first column specifies the considered instance ;
- The second column shows the optimal result of the $LR\bar{F}$;
- The third column gives the iteration at which the former value is reached: the following iterations do not decrease the former value ;
- The fourth column shows the total number of iterations of the algorithm ;
- The fifth column gives the final number of plans that have been computed at the end ;
- The sixth column gives the final number of constraints `[CT14-21]` that have been added to the MIP ;
- The seventh column gives the value of MIP when it is called at the end of the column generation process ;
- The eigth column gives the value of LP_R ;
- The ninth column gives the result of z defined in equation (A.93) ;
- The last column gives the elapsed time for the whole algorithm

The value of the relaxation of the master problem decreases as negative reduced cost columns are added. As the lowest value is reached in a few dozens of iterations and the columns added afterwards do not change this value, the optimal value of the restricted linear $\bar{F}$ ($RL\bar{F}$) is reached then. The number of columns generated is smaller than $|I|x$Tot It and as expected it decreases with the number of iterations.

Data	$RL\bar{F}$	It to LB	Tot It	Pl Nb	Cs Nb	MIP	LP_R	z	Time (s)
A0	8.73548e+12	4	24	32	40	8.73696e+12	8.73696e+12	8.73696e+12	0
A1	1.52339e+11	8	149	540	1106	1.52489e+11	1.68891e+11	1.69978e+11	11
A2	1.45276e+11	18	449	2111	1947	1.45745e+11	1.45747e+11	1.46435e+11	224
A3	1.52835e+11	8	764	1 929	1753	1.53190e+11	1.54055e+11	1.54747e+11	34
A4	1.02524e+11	18	461	2617	2626	1.02974e+11	1.11857e+11	1.12558e+11	157
A5	1.19655e+11	18	521	3226	3020	1.20145e+11	1.26698e+11	1.27744e+11	165
B6	7.75175e+10	22	761	6642	5630	7.90051e+10	8.16399e+10	8.48905e+10	287
B7	7.48221e+10	348	1094	8437	6105	7.57638e+10	7.87608e+10	8.21897e+10	896
B8	7.35414e+10	22	406	3256	7497	7.60027e+10	7.95534e+10	8.32061e+10	366
B9	7.31604e+10	28	372	3630	7477	7.56930e+10	7.96609e+10	8.34030e+10	512
B10	7.03327e+10	36	3075	11157	5944	7.13487e+10	7.52558e+10	7.90224e+10	1053
X11	7.34759e+10	40	1239	9061	5719	7.45279e+10	7.71169e+10	8.00708e+10	1533
X12	7.17523e+10	44	1958	10869	5640	7.22878e+10	7.49525e+10	7.82294e+10	737
X13	6.98451e+10	34	409	6299	6687	7.07385e+10	7.39050e+10	7.69910e+10	501
X14	6.92277e+10	40	611	7961	6911	6.99999e+10	7.36125e+10	7.69212e+10	1112
X15	6.73328e+10	44	935	10026	5955	6.78149e+10	7.13536e+10	7.50613e+10	676

Table A.2: Performance of the overall method with $\rho = 0.9$

Data	$RL\bar{F}$	It to LB	Tot It	Pl Nb	Cs Nb	MIP	LP_R	z	Time (s)
A0	8.73548e+12	4	24	32	40	8.73698e+12	8.73698e+12	8.73698e+12	0
A1	1.52342e+11	8	58	269	1072	1.52536e+11	1.68836e+11	1.69962e+11	7
A2	1.45284e+11	20	391	1614	1931	1.45769e+11	1.46151e+11	1.46840e+11	286
A3	1.52830e+11	8	776	1735	1694	1.53188e+11	1.54089e+11	1.54777e+11	33
A4	1.02543e+11	22	625	2687	2632	1.03002e+11	1.12506e+11	1.13198e+11	217
A5	1.19685e+11	18	367	2903	3059	1.20247e+11	1.27393e+11	1.28352e+11	173
B6	7.75271e+10	22	740	6283	5637	7.88730e+10	8.15095e+10	8.47410e+10	465
B7	7.48317e+10	38	661	9574	6130	7.55816e+10	7.85357e+10	8.19583e+10	936
B8	7.36053e+10	20	350	3311	7401	7.76852e+10	8.13098e+10	8.51822e+10	355
B9	7.31967e+10	30	1419	6187	7544	7.53907e+10	7.90747e+10	8.27811e+10	607
B10	7.03706e+10	34	3320	11417	5928	7.13445e+10	7.51940e+10	7.89976e+10	762
X11	7.34802e+10	38	1232	9508	5690	7.45335e+10	7.70939e+10	8.00418e+10	1563
X12	7.17785e+10	50	975	10821	5687	7.22479e+10	7.48730e+10	7.81379e+10	905
X13	6.98541e+10	24	535	6529	6667	7.08168e+10	7.38722e+10	7.68983e+10	575
X14	6.92260e+10	36	713	8243	6870	6.99515e+10	7.36255e+10	7.69829e+10	1122
X15	6.73713e+10	40	1032	11123	5977	6.78117e+10	7.12779e+10	7.49589e+10	690

Table A.3: Performance of the overall method with $\rho = 3$

Note that the number of lines (Cs Nb) and variables (Pl Nb) has the same order of magnitude. It means that both the rows and columns generation processes are necessary to find quickly the integer solution of the $RL\bar{F}$.

The value of $RL\bar{F}$ is always lower of the value of MIP (i.e., $R\bar{F}$) as expected since $z(RL\bar{F}) \leq z(R\bar{F})$, but it is not true that $z(RL\bar{F}) \leq z(F) \leq z$ since it is not a lower bound for the original problem.

Nevertheless, it interesting to note that the value of the objective function reported in column MIP is always smaller than the one reported in column LP_R which is always smaller than the value reported in the column z. It is clear that this is due to the approximations that have been done for solving $\bar{F}$. The increase in the objective function between MIP and (LP_R) can be explained by the fact that MIP uses an average scenario instead of the whole set of scenarios. Therefore, when problems (LP_R) are solved with the dates of outage and `BO` fixed, can happen that for satisfying scenarios with a demand greater than the average it is necessary to use more expensive source of energy.

Less intuitive is the explanation of why the values reported in LP_R is smaller than z. This difference is due to the modulation. In fact, when the model considers the demand aggregated over a week it applies a certain quantity of modulation. Nevertheless, can happen that when the time interval considered is the time step it is impossible to maintain the same modulation and therefore more expensive source of energy must be used for satisfying the excess of demand.

Table A.4 gives CPU time in seconds of the method:

- The first column specifies the considered instance ;
- The second column shows the time to read the input and to write the output ;
- The third column gives the time to build the graph for the pricing problem ;
- The fourth column shows the total time of column generation ;
- The fifth column gives the time to solve the integer master problem obtained at the end of the column generation process ;
- The sixth column gives the time of the reload phase ;
- The seventh column gives the time of the production phase ;
- The last column gives the elapsed time for the whole algorithm

Actually the whole method never reaches the set time limit (3600 seconds). It can be used, for example, as a very good initial solution which could be a starting point for a local search procedure. The most time-consuming phase is the MIP phase at the end of the column generation process. The reload phase is more time consuming than the production phase due to the

Data	Read/Write	Graph	CG	MIP	Reload	Production	Total
A0	0	0	0	0	0	0	0
A1	1	0	1	5	3	1	11
A2	4	173	15	12	19	1	224
A3	3	1	6	8	15	1	34
A4	8	95	12	12	28	2	157
A5	7	26	76	25	28	3	165
B6	56	5	27	93	94	12	287
B7	52	18	122	543	152	9	896
B8	60	48	94	101	45	18	366
B9	143	66	164	52	65	22	512
B10	136	8	104	576	209	20	1053
X11	48	537	152	577	199	10	1533
X12	53	11	76	434	155	8	737
X13	143	126	57	102	42	31	501
X14	139	76	129	643	104	21	1112
X15	132	8	54	193	267	22	676

Table A.4: CPU Time (s)

multiple resolutions for the different combinations of columns that produce the MIP. We can notice that the time taken by the input and output is not negligible due to the large size of the files to read and write.

A.10 Conclusion

We presented a mathematical formulation for the EDF (Électricité De France) electricity production planning problem submitted as subject of the ROADEF/EURO 2010 Challenge, and a heuristic method based on column generation. To the best of our knowledge this is the first ranked method using mathematical formulation among the final results of the Challenge. The method proceeds in two stages : first we fix dates for outages for each nuclear plant on aggregated data, and in a second step we determine the reload and production quantities for each plant and each scenario. Computational results show the effectiveness of the proposed approach, given that the method was far from using all the time allowed in the Challenge. Indeed, part of the time is kept for solving the (LP_R) and the (LP_s) problems which may be time consuming. Thus, the first stage of the method is stopped after a time limit which has been experimentally set to 30 minutes. Nevertheless, in some case it stops before, since no more columns are produced.

Amongst possible future improvements of the proposed approach and therefore future directions of research, it could be interesting, once a real lower bound is obtained, to add more positive reduced cost columns which are in the gap. The current shortest path solving algorithm is not suited to that, since it produces columns with minimal reduced costs. A further improvement could consist in generating more sets of schedules for the outages using the local search on the set of outage dates found in the first phase of the method. Last but not least, it would be interesting to improve the method by entering the clique-like constraints (A.36) into the pricing problem. Therefore the associated dual variables could be included in the pricing subproblem. Introducing them could improve the number of columns found by the current shortest path method and alleviate the need for our diversification algorithm.

Acknowledgment

We wish to thank anonymous reviewers for fruitful suggestions which help improve a previous version of this paper. Moreover, we would like to thank Aristide Mingozzi and Guillaume Turri for their useful help and/or comments they gave at the beginning of this project.

Appendix B

Challenge ROADEF 2012: Machine Assignment Problem

This work has been done together with Bernat Gacias and Paolo Gianessi.

Introduction

More and more information is available on the web. To help users finding the piece of information they need, several web search engines were created. The competition between them is tough as users want both accurate and quick results. A wide field of study concerns the design of algorithms able of promptly providing relevant results to users. The quickness of engines based on these methods relies on their design, but also on the availability of resources to assign them to. This need for huge amounts of resources – such as CPU or RAM – leads to the construction of computer clusters; but then, finding a good assignment of all the tasks to the different available machines is a key factor to improve the overall efficiency. Finding an optimal assignment in such a context is the subject of the challenge proposed by Google for the next ROADEF/EURO Conference. Its whole description can be found in [1].

An instance of the proposed problem is made up by a set of processes, each one with specific resources consumption profiles. A set of machine is given with their resources capacities. An initial feasible assignment is provided. Processes can be moved from their initial assignment to a different machine leading to a new cost for the solution. However these moves must respect numerous hard constraints. Some constraints can be independently checked such as capacity constraints, but others link all the processes together in case of dependency between

processes or restrictions due to potential conflicts. The new processes assignment must minimize a given cost function, which depends on machines load, resources usage balance and process displacement costs.

In the following, we briefly describe the method used to solve the problem. Section B.1 presents the MILP program that was used to model the problem; Section B.2 describes the method used; Section B.3 gives the results obtained on the two sets of instances that were released by the organizers.

B.1 The model

In this section, we give a MILP program which models the problem. The notations that are used are the same as those introduced in in [1]. Otherwise they are introduced in the following subsections. The model is then given in two parts: first the objective function that is composed by five different costs; then the constraints. The constraints are of two kinds: those given by the problem and others logical constraints.

Parameters

- $n_m = |\mathcal{M}|$: number of machines
- n_s: number of processes per service $s \in \mathcal{S}$
- n_l: number of machines per location $l \in \mathcal{L}$

Variables

- x_{pm}: binary variable indicating if process $p \in \mathcal{P}$ is assigned to machine $m \in \mathcal{M}$
- y_{sl}: binary variable indicating if service $s \in \mathcal{S}$ is present at location $l \in \mathcal{L}$
- z_{rm}: integer variable measuring the excess of the safety capacity of resource $r \in \mathcal{R}$ at machine $m \in \mathcal{M}$
- t_{bm}: integer variable computing the result of the balance cost triple $b \in \mathcal{B}$ at machine $m \in \mathcal{M}$
- $smcVar$: integer variable measuring the service move cost

Objective function

$$
\begin{aligned}
totalCost &= \sum_{r\in\mathcal{R}} weight_{loadCost}(r)\cdot(\sum_{m\in\mathcal{M}} z_{rm}) \\
&+ \sum_{b\in\mathcal{B}} weight_{balanceCost}(b)\cdot(\sum_{m\in\mathcal{M}} t_{bm}) \\
&+ \sum_{p\in\mathcal{P}}\sum_{m\in\mathcal{M}} x^0_{mp}\cdot(1-x_{mp})\cdot PMC(p)\cdot weight_{processMoveCost} \\
&+ smcVar\cdot weight_{serviceMoveCost} \\
&+ \sum_{p\in\mathcal{P}}\sum_{(m_1,m_2)\in\mathcal{M}^2, m_1\neq m_2} MMC(m_1,m_2)x^0_{m_1p}x_{m_2p}\cdot weight_{machineMoveCost}
\end{aligned}
$$

Constraints

$$
\begin{aligned}
\sum_{p\in\mathcal{P}} x_{pm}R(p,r) &\leq C(m,r) && \forall m\in\mathcal{M}, r\in\mathcal{R} && (i)\\
\sum_{p\in s} x_{pm} &\leq 1 && \forall m\in\mathcal{M}, s\in\mathcal{S} && (ii)\\
\sum_{l\in\mathcal{L}} y_{sl} &\geq spreadMin(s) && \forall s\in\mathcal{S} && (iii_a)\\
y_{sl} &\geq \frac{\sum_{p\in s, m\in l} x_{pm}}{\min\{n_m, n_s, n_l\}} && \forall s\in\mathcal{S}, l\in\mathcal{L} && (iii_b)\\
y_{sl} &\leq \sum_{p\in s, m\in l} x_{pm} && \forall s\in\mathcal{S}, l\in\mathcal{L} && (iii_c)\\
y_{sl} &\in \{0,1\} && \forall s\in S, l\in L && (iii_d)\\
\sum_{p^a\in s^a}\sum_{m\in n} x_{p^am} - n_{s^a}\sum_{p^b\in s^b}\sum_{m\in n} x_{p^bm} &\leq 0 && \forall n\in\mathcal{N}, p^a\in s^a && \\
& && s^a \text{ dep. on } s^b && (iv)\\
\sum_{p\in\mathcal{P}}(x^0_{pm}+(1-x^0_{pm})\cdot x_{pm})\cdot R(p,r) &\leq C(m,r) && \forall m\in\mathcal{M}, r\in\mathcal{R} && (v)
\end{aligned}
$$

Constraints (i) ensure the respect of the capacity constraints. Constraints (ii) assure that there is no two processes of the same service at the same machine. Constraints (iii_a) to (iii_d) assure the respect of the spread constraints. Constraints (iv) stand for the dependency constraints. Constraints (v) ensure the transient usage constraints. Additional logical constraints have to be

added:

$$
\begin{aligned}
z_{rm} &\geq 0 && \forall r \in \mathcal{R}, m \in \mathcal{M} \\
z_{rm} &\geq \sum_{p \in \mathcal{P}} x_{pm} \cdot R(p,r) - SC(m,r) && \forall r \in \mathcal{R}, m \in \mathcal{M} \\
t_{bm} &\geq 0 && \forall b \in \mathcal{B}, \in \mathcal{M} \\
t_{bm} &\geq target \cdot \Bigg(\Big(C(m,r_1) - C(m,r_2) \Big) \\
&\quad - \Big(\sum_{p \in \mathcal{P}} x_{pm} \cdot (R(p,r_1) - R(p,r_2)) \Big) \Bigg) && \forall b \in \mathcal{B}, m \in \mathcal{M} \\
smcVar &\geq 0 \\
smcVar &\geq \sum_{m \in \mathcal{M}, p \in s} x^0_{mp} \cdot (1 - x_{mp}) && \forall s \in \mathcal{S}
\end{aligned}
$$

B.2 The method

The former model is an exact model. However it is not tractable to run it within the time limit imposed by the challenge, even on the small instances. To obtain good solutions, three different methods are used. The third method is designed to cope with the B instances, which size exceeds $5'000$ processes.

The local search The local search is based on two simple moves: *shift* and *swap*. Both moves browse all processes and/or machines. For the B instances, only a randomly chosen subset of processes or machines are browsed because of their size. Note here that because of transient usage constraints, some processes cannot be assigned to some machines.

With the shift move, we try to move each process $p \in \mathcal{P}$ to another machine $m \in \mathcal{M}, m \neq M(p)$, where $M(p)$ is its current assignment. If this move improves the objective function cost and is valid, it is kept in memory. Once all the machines are tested, the best move for the process is kept in a stack of promising moves. The size of the stack is limited by a value Nb_{shift} keeping only the best processes to move sorted in a decreasing order of the gain in the cost function.

With the swap move, we try to swap a process $p \in \mathcal{P}$ with another process $p' \in \mathcal{P}$ such that $M(p') \neq M(p)$. If this move improves the objective function cost and is valid, then it is kept in memory. Here also once all the processes p' are tested, the best swap for p is kept in a stack of promising moves. The size of the stack is limited by a value Nb_{swap} keeping only the best processes to swap, sorted again in a decreasing order of the gain in the cost function.

In both cases, once all the promising moves are found, we build a solution with all these moves sequentially realized. We start from the first one of the stack. Then, following their position in the stack, the others are tried. For each move in the stack, if the new solution is still valid and improves the solution, the move is then performed.

The MILP driven search on increasing solution space The solution obtained by the local search is then used to provide a warm start to the MILP model to further improve the cost of the new process assignment. However, since solving the exact model would be too time consuming, only a small model is built. To each machine is associated the sum of its balance cost and its load cost. Then only the $2m$ machines having the lowest and the greatest costs are selected. The processes that are assigned to them can be moved, while the others are frozen at their current machine. If time enables, the solver is run several times increasing m and the size of the model. Between two calls to this method, the former local searches are performed for 10 iterations on the solution obtained.

The Super Process: aggregating processes Although the former method enables to deal with the $A1$ and $A2$ instances, the B instances size does not allow to consider all the processes while calling the model. For that purpose, processes are aggregated to form *super processes*. These structures are then used in the MILP program rewritten to use super process instead of process as variables. This reduces the number of variables and so the new MILP program can be solved. These super processes have for resource consumption the sum of the resource consumptions of the processes it represents, and gather several services. The combination of services enables to solve some of the dependency constraints. The creation of these super process changes from a call to another, in order to enable diversity. The number of processes in a super process is between 1 and 20, depending on the size of the instances. There are created by taking into account the repartition of the process services in the current solution. When a super process is moved from a machine to another, all the processes it represents are moved.

B.3 The results

Here we give the results. The machine used has a Pentium(R) Dual-Core E5500 2.80GHz with 3.8Go RAM on Debian version 64 version 7. The parameters Nb_{shift} is set to 50 and Nb_{swap} to the number of processes. The initial local search is run for 100 seconds, then the

MILP driven search is called, possibly several times. When the remaining time is less than 60 seconds before the end, the local search is run starting from the best solution encountered.

Instance	Initial solution	Final solution	Deviation (%)	CPU time (s)
a1_1	49528750	44306501	89.4561	0.12
a1_2	1061649570	780581762	73.5254	489.32
a1_3	583662270	583006017	99.8876	421.25
a1_4	632499600	274364352	43.3778	516.01
a1_5	782189690	727578309	93.0181	510.24
a2_1	391189190	4168765	1.0657	525.90
a2_2	1876768120	895761559	47.7289	526.38
a2_3	2272487840	1399146389	61.5689	492.88
a2_4	3223516130	1773997601	55.0330	530.36
a2_5	787355300	529941864	67.3066	554.12
b_1	7644173180	3554115209	46.4944	543.72
b_2	5181493830	1019623424	19.6782	554.68
b_3	6336834660	172170487	2.7170	512.84
b_4	9209576380	4677870607	50.7935	467.04
b_5	12426813010	931854930	7.4987	522.15
b_6	12749861240	9525873556	74.7174	517.03
b_7	37946901700	14990130791	39.5029	488.07
b_8	14068207250	1215576946	8.6406	508.28
b_9	23234641520	15885583344	68.3703	514.53
b_10	42220868760	18099987199	42.8698	470.32

Table B.1: Results on the released instances

Bibliography

[1] "Google ROADEF/EURO challenge 2011–2012: Machine reassignment", 2011.