Algorithmes d'optimisation pour un service de transport partagé à la demande

Sven Vallée

UNIVERSITÉ DE LORRAINE

Ecole doctorale IAEM Lorraine

Algorithmes d'optimisation pour un service de transport partagé à la demande

THÈSE

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Sven Vallée

Composition du jury :

Rapporteurs :	Pr. Olivier Péton	IMT Atlantique
	Pr. Nacima Labadie	Université de Technologie de Troyes
Examinateurs :	Pr. Aziz Moukrim	Université de Technologie de Compiègne
	Pr. Ye-Qiong Song	Université de Lorraine
	Pr. Ammar Oulamara	Université de Lorraine
	Mcf. Wahiba Ramdane-Cherif	Université de Lorraine
Invité :	M. Samir Naim	Padam Mobility

Laboratoire Lorrain de Recherche en Informatique et ses Applications
UMR 7503

Table des matières

Chapitre 1

Introduction générale

1.1 Introduction

De nombreuses grandes villes à travers le monde font face à des problèmes de transport comme la congestion du trafic et la consommation d'énergie essentiellement d'origine fossile. Ceci implique une augmentation constante des émissions de dioxyde de carbone et de particules fines, engendrant des conséquences néfastes sur l'environnement et la santé des citoyens. L'activité de transport est responsable d'une grande partie des émissions de dioxyde de carbone. En France par exemple, elle représente 38% des émissions totales [1] dont 74% sont dues aux camions, bus et voitures individuelles [2]. En milieu urbain, le taux d'occupation moyen des véhicules individuels est de 1,06 avec une majorité de déplacements effectués uniquement avec le conducteur à bord. Il existe donc une marge importante de réduction des coûts et de la pollution grâce à la seule élaboration d'incitations pour une meilleure occupation des véhicules individuels. De nombreux nouveaux défis sont donc ouvert pour réduire l'importance des véhicules individuels dans la chaîne de mobilité et pour une utilisation plus intelligente des véhicules.

De nombreuses nouvelles initiatives ont vu le jour ces dernières années en complément du transport en commun traditionnel. Le service de transport à la demande en mode partagé, qui est à mi-chemin entre le taxi et le bus, constitue une bonne alternative au véhicule individuel. La flexibilité des horaires et la souplesse des itinéraires permettent d'éviter de nombreux trajets à vide. Par ailleurs, le développement des nouvelles technologies d'information et de communication (TIC) durant les dix dernières années a largement accéléré l'émergence de nouveaux services de transport à la demande. Ce dernier s'est ainsi généralisé à un large public pour de nombreux motifs de déplacements. La mise en oeuvre d'un tel système de transport à la demande partagé dynamique et offrant une qualité de service élevée constitue une vraie innovation. Un des défis d'un tel système est le développement de la couche décisionnelle constituée d'algorithmes d'optimisation efficaces permettant de proposer une réponse rapide en temps réel et adaptée aux caractéristiques de chaque demande.

1.2 Objectif de la thèse et organisation du manuscrit

L'objectif de cette thèse CIFRE est le développement de méthodes d'optimisation pour le déploiement d'un système de transport à la demande dynamique et partagé. La thèse se déroule

1. Chiffres Citepa pour l'année 2017
2. Chiffres AIE pour l'année 2016

en partenariat avec la start-up parisienne Padam [3]. L'ensemble des outils développés sont testés et validés sur la plateforme de transport de l'entreprise. La suite du manuscrit est organisée comme suit :

Dans le chapitre 2, nous présenterons l'entreprise Padam et le contexte industriel sous-jacent. Nous parlerons des produits proposés par l'entreprise et montrerons comment la problématique d'optimisation constitue le coeur de son savoir-faire.

Le chapitre 3 présentera un état de l'art sur le DARP (Dial-A-Ride Problems) qui est le modèle de la littérature le plus proche des problématiques de l'entreprise. Nous mettrons en lumière les principaux manques de la littérature nécessitant le développement de nouvelles techniques d'optimisation pour traiter au mieux les spécificités posées par le service de transport opéré par Padam. Nous présenterons également la formalisation mathématique à la base des problématiques rencontrées et les instances fournies par Padam qui serviront de base aux expérimentations de la thèse.

Le chapitre 4 présentera le module online, notre première contribution dans le système d'optimisation de l'entreprise. Ce module constitue le levier d'optimisation principal de l'entreprise et permet de traiter les requêtes des utilisateurs en temps réel. Nous présenterons les innovations apportées comblant les manques de la littérature et les expérimentations menées sur les instances de l'entreprise.

Dans le chapitre 5, nous étudierons un sujet quasi inexistant dans la littérature scientifique : la réinsertion en temps réel des requêtes ne pouvant être insérées par le module online. Nous présenterons le module de réinsertion que nous avons développé, permettant de traiter en temps réel les requêtes rejetées par le module online. Nous étudierons et comparerons deux nouvelles heuristiques par le biais d'expérimentations sur les instances de l'entreprise.

Le chapitre 6 est dédié à un autre sujet peu traité dans la littérature : le développement d'un module offline permettant d'optimiser les trajets des véhicules entre les réservations des utilisateurs. Nous présenterons en détail l'heuristique utilisée et le réglage des hyper-paramètres. L'étude de l'impact d'un tel module sur les performances du système sera réalisée avec les instances fournies par Padam.

Le chapitre 7 se concentrera sur l'intégration des différents modules proposés dans la plateforme d'optimisation de Padam. Nous expliquerons les difficultés et la façon dont nous avons transformé les algorithmes proposés en des modules d'optimisation souples et robustes.

Enfin, le chapitre 8 contient les conclusions générales sur le travail réalisé durant la thèse et les perspectives d'amélioration.

3. www.padam.io

Chapitre 2

Présentation du contexte et de la problématique

2.1 Introduction

Dans ce chapitre, nous commencerons par introduire en section 2.2 le contexte de mobilité dans lequel s'inscrit la thèse puis nous présenterons en section 2.3 l'entreprise et le produit qu'elle propose. Nous parlerons ensuite du moteur d'optimisation actuel en section 2.4, ce qui nous amènera naturellement à la problématique de la thèse présentée section 2.5.

2.2 Contexte de la mobilité

Selon les chiffres Citepa pour l'année 2017, l'activité de transport représente 38 % de l'émission totale de dioxyde de carbone en France. Cette dernière est ainsi la principale source d'émission de dioxyde de carbone devant les résidences et l'industrie. Concernant l'activité de transport, près de trois quarts (74.2%) des émissions sont dues à des transports routiers. En outre, la plupart des déplacements en véhicule individuel ont été effectués avec seulement le conducteur à bord du véhicule. La situation est similaire dans le monde : selon [Stephan, 2007], le coût des sièges vides dans le déplacement des véhicules est estimé à 500 milliards d'euros par an sur toute la planète. Le domaine du transport est ainsi devenu un réel enjeu sur les plans économique et écologique auquel les collectivités et les transporteurs publiques et privés doivent faire face. Au delà d'une simple incitation à une meilleure occupation des véhicules individuels, de nouveaux défis et initiatives se présentent pour réduire l'importance des véhicules individuels dans la chaîne de mobilité et pour une utilisation intelligente des véhicules.

Dans cette perspective de nouvelles mobilités et fort de l'essor récent des systèmes de communication, plusieurs initiatives d'auto-partage et de covoiturage ont vu le jour. Citons par exemple Blablacar, UberX, Via ou encore AutoLib. Ces initiatives constituent une réponse aux défis économiques et écologiques pré-cités et amorcent le changement de paradigme où l'utilisateur cherche davantage un service de mobilité qu'un moyen de transport personnel. Ces nouveaux services offrent à l'utilisateur une grande flexibilité à un tarif compétitif en proposant dans certains cas une alternative solide aux systèmes de transport en commun classiques. Ces derniers, de leur côté, ont en effet très peu bénéficié de l'avènement des nouvelles technologies. C'est le cas par exemple dans les territoires à faible densité de population où les services de bus proposés sont souvent coûteux pour les transporteurs et peu flexibles pour l'utilisateur.

Le transport en commun à la demande (TCAD) répond à ces enjeux en livrant un niveau de service proche de ce que font les VTC (Voiture de Transport avec Chauffeur) dans les grandes villes, à un coût bien inférieur et avec un impact environnemental beaucoup moins important. Il constitue une excellente alternative au véhicule individuel de par son coût réduit tout en offrant une plus grande flexibilité que le transport en commun classique.
L'innovation en matière de TCAD ayant été concentrée essentiellement sur les zones à forte densité de population (essentiellement des grandes villes), les territoires et agglomérations sont demandeurs de telles avancées en complément voir en remplacement de leur système de transport en commun actuel. Il existe ainsi de nombreuses opportunités pour développer des services de TCAD efficaces et faciles à utiliser. Les entreprises qui s'attaquent à cet Everest font face à un enjeu d'interconnexion et d'intermodalité : ces dernières doivent pouvoir en effet s'interfacer entre elles pour offrir une expérience unifiée à la fois aux élus (dans le cadre du transport public), aux autorités organisatrices de transport, et enfin aux habitants. La start-up *Padam* s'inscrit dans ce contexte en proposant des logiciels visant à optimiser les systèmes de transport en commun à la demande. L'entreprise se positionne en tant que fournisseur de Software as a Service (SaaS) et propose à ses clients de payer une mensualité pour déployer et utiliser le progiciel Padam sur les territoires souhaités. La section suivante décrit plus en détails l'entreprise.

2.3 Présentation de l'entreprise Padam

2.3.1 Un peu d'historique

Padam est lancée en mai 2014 par deux étudiants de Polytechnique, Ziad Khoury et Grégoire Bonnat, après une phase de recherche et d'étude de 6 mois. L'objectif de l'entreprise : repenser le transport en commun pour le rendre plus à l'écoute de l'utilisateur et plus attentif à son confort. La préoccupation à l'origine de l'initiative : comment rentrer chez soi la nuit à Paris ? En effet, le métro est fermé après une certaine heure et le service offert par le Noctilien en Île de France (bus de nuit) est restreint. Pour les nombreuses personnes qui n'utilisent pas leur voiture personnelle à Paris, deux solutions se présentent : rentrer à pieds ou prendre un taxi/VTC. L'idée à l'origine de Padam est donc la suivante : proposer un service flexible et peu coûteux à mi-chemin entre le taxi et le bus. En juin 2014 naît ainsi *Padam Night*, un service de transport en commun à la demande où les utilisateurs peuvent réserver en avance ou en temps réel un trajet via une application mobile. A ses débuts, le service est événementiel avec un unique point de prise en charge : les clients commandent un Padam à la sortie d'un concert (par exemple) en spécifiant leur point de dépose à l'intérieur de Paris. Le service s'ouvre ensuite progressivement les nuits du week-end (vendredi soir et samedi soir) avec une prise en charge/dépose dans tout Paris intra-muros. La mise en oeuvre et l'originalité de ce service permettent à Padam de gagner plusieurs prix et de lever la somme de 500 000 euros en septembre 2015. Cependant, après avoir transporté des centaines de personnes, le service s'arrête en janvier 2016 à cause d'un coût de fonctionnement trop élevé. Fort de cette expérience enrichissante, l'entreprise décide d'attaquer un marché peu exploré jusqu'alors : le "commuting", consistant à transporter les utilisateurs entre leur domicile et leur travail. Le service *Padam Daily* est ainsi lancé en mai 2016. Il permet à toutes les personnes habitant et travaillant à l'ouest de Paris de bénéficier d'un service rapide et confortable pour se rendre à leur travail, tout en proposant des tarifs inférieurs à ceux d'un VTC classique. *Padam Daily* est ouvert du lundi au vendredi entre 7h30 et 9h30 le matin et entre 18h et 20h le soir. Cependant, ce dernier rencontrera un engouement moins prononcé que *Padam Night* et s'arrêtera à la fin de la même année, en décembre 2016. Les arrêts de *Padam Night* et *Padam Daily* amènent rapidement au constat suivant : exister face à la concurrence

sur le marché du transport à la demande (TAD) en B2C (Business to Customer) demande des moyens financiers très importants dont ne dispose pas encore l'entreprise. Il devient donc urgent de faire "pivoter" la stratégie commerciale. C'est un autre service ouvert par Padam qui va aiguiller ce changement de cap : *Slide*. Inspiré par *Padam Daily*, *Slide* est officiellement ouvert en juillet 2016 sur la ville de Bristol en Angleterre. A la différence des deux autres services de l'entreprise, ce dernier est fait en partenariat avec un opérateur de transport, RATPDev (filiale de la RATP). Padam fournit la technologie (logiciel, applications mobiles) sur la base d'une rémunération mensuelle à l'opérateur de transport qui se charge de fournir une flotte de véhicules, gère la relation client et supervise les opérations de marketing. Constatant l'efficacité de ce mode de fonctionnement et poussé par la difficulté du B2C, les deux fondateurs décident de changer la stratégie économique pour s'orienter vers du B2B (Business to Business). A l'aube de l'année 2017, Padam change son positionnement et devient fournisseur de Software as a Service (SaaS). Le client cible n'est désormais plus l'utilisateur final mais les différents opérateurs de transport tels que RATP, Transdev, Keolis et/ou directement les communes et communautés de communes. A l'instar de *Slide*, Padam propose à ces derniers de payer une mensualité pour avoir la possibilité d'utiliser l'ensemble de la solution logicielle et de bénéficier des mises à jour régulières. La prochaine section présente plus en détails les différentes composantes de l'offre générale de Padam.

2.3.2 Le produit Padam

Le produit Padam se décline en deux composantes principales : *PadamLive* et *PadamLab*.

PadamLab

PadamLab est un outil d'aide à la décision pour TCAD permettant de créer et d'optimiser des services adaptés au contexte local du transporteur. Les études d'aide à la décision sont vendues à la carte avec une tarification sur-mesure. Une étude vise à anticiper les performances d'un service de TCAD sans avoir à l'opérer réellement. L'étude fournit les résultats détaillés (itinéraires des véhicules, expérience de l'utilisateur) des différents scénarios de dimensionnement des services. Les résultats sont extraits, interprétés et envoyés au client. L'image 2.1 résume le fonctionnement de *PadamLab*.

Figure 2.1 – Illustration de *PadamLab*.

PadamLive

PadamLive est une solution clé en main de réservation et d'optimisation sous licence logiciel SaaS permettant aux transporteurs de déployer rapidement un système de TCAD sur leur territoire avec des coûts maîtrisés. Elle contient :

— Deux applications mobiles (iOS et Android) destinées aux utilisateurs finaux. Elles permettent à ces derniers de réserver leur trajet en avance ou en temps réel, d'être notifié des éventuels changements d'horaire, d'entrer leurs favoris, de gérer les différentes réservations etc... Les applications sont proposées en marque blanche et sont ensuite personnalisées selon la charte graphique du transporteur.
— Un site web de réservation remplissant le même objectif que les applications mobiles.
— Deux applications mobiles (iOS et Android) destinées aux chauffeurs des véhicules. Elles permettent à ces derniers de connaître leur trajet en temps réel, d'être notifiés d'éventuelles déviations, d'être dirigé vers des parkings etc... Ces applications sont également proposées en marque blanche.
— Une application web d'administrateur permettant aux gestionnaires de créer des services pour chaque journée, de suivre le déplacement des véhicules, de gérer les éventuels litiges avec les clients, d'obtenir les statistiques des services passés etc...
— Une interface pour centre d'appel, utilisée par les transporteurs qui mettent un centre d'appel à la disposition de leurs utilisateurs. Elle permet facilement à l'opérateur de faire des réservations en réponse aux appels des clients.
— Un moteur d'optimisation permettant de construire les itinéraires des véhicules en fonction des requêtes client. La section suivante présente plus en détails son fonctionnement.

L'image 2.2 présente la vitrine commerciale de *PadamLive*.

Figure 2.2 – Vitrine commerciale de *PadamLive*.

2.4 Moteur d'optimisation actuel

Lorsqu'un transporteur souscrit à l'offre *PadamLive*, il est en mesure de créer des services sur son territoire. Un service est une plage horaire sur laquelle des véhicules sont mis à disposition pour servir des utilisateurs. Chaque véhicule est disponible sur une plage horaire qui lui est propre à l'intérieur du service. Il peut y avoir plusieurs services dans une même journée. Dans le cadre d'une offre de "commuting" par exemple, il peut y avoir un service pour le matin et un pour le soir. Dans un cadre d'un transport public, il peut y avoir un service de 12h sur la journée, de 8h à 20h par exemple. Pour ouvrir un service sur un territoire, il est nécessaire de définir au préalable un maillage. Ce dernier est un ensemble de points géographiques qui vont constituer les points de prise en charge et de dépose des clients ainsi que les lieux de stationnement des véhicules. Chaque point de ce maillage est appelé un noeud. Les noeuds

sont placés dans des endroits stratégiques et peuvent correspondre ou non à des arrêts de bus physiques. Dans le cas de *Slide* par exemple, le maillage comporte plus de 300 noeuds répartis sur toute la ville de Bristol. Le maillage est déterminé principalement par le transporteur grâce à sa connaissance du terrain. Les distances et temps de trajet entre tous les noeuds sont calculés à partir de données ouvertes.

Lorsqu'un service est ouvert, les habitants du territoire peuvent réserver des trajets via une des deux applications mobiles ou par le site web (et le centre d'appel dans certains cas). Pour réserver un trajet, un utilisateur renseigne ses adresses de départ et d'arrivée, le nombre de passagers demandé et l'heure de prise en charge souhaitée (le chapitre 4 présente plus en détails le processus de réservation). Les noeuds les plus proches des adresses indiquées lui sont communiqués ainsi que le temps de marche à ces noeuds depuis ces mêmes adresses. On appelle noeud de pickup le noeud associé à l'adresse de départ de l'utilisateur et noeud de dropoff le noeud associé à son adresse d'arrivée. Le client devra se rendre à son noeud de pickup pour être pris en charge et sera déposé à son noeud de dropoff. Le service proposé par Padam n'est donc pas un service de porte à porte. Cette particularité permet de mutualiser plus facilement les clients allant dans la même direction et est indispensable dans le cadre du transport public.

Les utilisateurs peuvent réserver un trajet à l'avance ou en temps réel. Dans les deux cas, l'utilisateur reçoit une réponse à sa demande en l'espace d'une ou deux secondes. La demande peut être refusée, dans le cas où les véhicules sont déjà remplis par exemple. Si le client reçoit une proposition du système, il est libre de la refuser. Dans le cas contraire, il est inséré dans un véhicule et des heures de prise en charge et de dépose lui sont communiquées. Le système rajoute des fenêtres horaires à ces heures pour assurer que les insertions des futurs utilisateurs ne vont pas perturber de manière significative les horaires communiqués. Un temps de trajet maximum est également imposé sur chaque requête pour assurer que l'utilisateur ne subira pas un détour trop important sur son trajet. Chaque requête acceptée est ainsi associée à des contraintes temporelles permettant de maintenir une qualité de service élevée à tous les utilisateurs. En plus de devoir fournir une réponse rapide, le système doit également créer des tournées de haute qualité pour servir un maximum d'utilisateurs sur la durée du service et minimiser les déplacement des véhicules. Tout en cherchant à respecter ces impératifs, le système doit veiller à ne pas violer les contraintes horaires pré-citées et les contraintes de capacité et de disponibilité des véhicules.

Lorsqu'un client fait une demande de trajet, Padam utilise une heuristique simple et rapide pour apporter une réponse. La tâche de cette heuristique est fondamentale : construire et optimiser les tournées. Elle constitue l'unique système d'optimisation et le seul levier d'action pour optimiser le transport à la demande chez Padam.

2.5 Problématique de la thèse

Comme l'a montré la section précédente, le fonctionnement des services opérés par Padam engendre naturellement le besoin d'avoir à disposition un module d'optimisation à la fois interactif, flexible et rapide permettant un choix multiple à l'utilisateur en l'espace d'une ou deux secondes. L'heuristique d'insertion utilisée par l'entreprise propose cependant un levier d'action très limité. En effet, les tournées sont créées requête par requête en fonction de l'ordre d'arrivée ce qui rend l'optimisation myope et de plus en plus éloignée de l'optimum au fur et à mesure de l'augmentation du nombre d'utilisateurs. De plus, une requête est systématiquement

rejetée lorsqu'aucune insertion faisable n'est trouvée, ce qui peut aboutir à un grand nombre d'utilisateurs rejetés par le système. L'heuristique d'insertion en elle-même est lente et contient plusieurs paramètres libres dont l'impact n'est pas étudié. Ces différentes limites suggèrent qu'il est possible d'apporter des améliorations importantes au système d'optimisation de l'entreprise. De plus, il existe un réel manque à combler dans la littérature sur les problèmes de "Dial-A-Ride" (DARP) dynamiques, qui est le modèle de base des problèmes de tournées de véhicules se rapprochant le plus du contexte de Padam. De manière générale, peu de travaux ont été mené sur le DARP dynamique et aucun de ces travaux ne permet de traiter toutes les particularités posées par le système de transport de Padam. Parmi les travaux considérant une heuristique d'insertion, certains ne considèrent qu'un seul véhicule ([Coslovich et al., 2006]), utilisent un nombre infini de véhicules ([Wong et al., 2014], [Häll et al., 2012]) ou considèrent des fenêtres temporelles souples pour les clients ([Beaudry et al., 2010]). D'autres travaux utilisant des méthodes d'optimisation en continue acceptent des temps de réponse client trop long ([Attanasio et al., 2004]) ou ne traitent pas les clients immédiatement après leur requête ([Santos and Xavier, 2015]). A notre connaissance, seul [Luo and Schonfeld, 2011] propose une heuristique où les requêtes rejetées sont réinsérées en temps réel. Cependant, la flotte considérée est infinie et l'heuristique proposée est sujette à de nombreuses améliorations. De plus, tous les travaux de la littérature portés à notre connaissance proposent une solution unique au client, laissant ainsi de côté la complexité liée à la possibilité de choix multiple. Le point de pickup/dropoff étant toujours celui du client ou le plus proche de son adresse, aucun travail sur l'optimisation des points de ramassage et dépose n'a à notre connaissance été mené dans la littérature sur le DARP dynamique.

L'objectif de la thèse est de concevoir et de développer des méthodes d'optimisation efficaces pour le DARP dynamique permettant de répondre aux défis exposés ci-dessus. Il s'agira d'améliorer l'heuristique existante et d'apporter des nouvelles briques d'optimisation pour rendre le système plus performant. L'originalité de l'approche consiste à proposer une solution globale avec trois modules d'optimisation, interagissant dans un système interactif et dynamique (cf figure 2.3). Chaque module permet de gérer une stratégie d'optimisation différente : le module online a pour but de répondre à l'utilisateur en quelques secondes tout au plus. Nous présenterons l'heuristique que nous avons implémentée et qui remplace l'heuristique actuellement utilisée par Padam. L'objectif sera d'étudier les différents paramètres assurant la cadence des propositions au client et d'analyser le bénéfice obtenu par l'intégration du choix des noeuds de prise en charge du client dans le processus d'optimisation. Le module de Réinsertion permet d'insérer en temps réel un client qui a reçu une réponse négative du module online. Nous présenterons deux heuristiques différentes et montrerons le gain apporté en terme de requêtes servies et de coût de transport. Enfin, le module offline permet d'optimiser les tournées pendant le temps disponible entre deux requêtes client. Nous présenterons l'heuristique utilisée et des simulations mettant en évidence son apport bénéfique sur les performances du système.

2.6 Conclusion

Le domaine du transport a subi de profondes transformations ces 10 dernières années. L'essor des technologies de communication a favorisé le développement de nouvelles offres de déplacement. Le TCAD devient ainsi une alternative de plus en plus considérée par les opérateurs de transport désireux de faire évoluer leur offre actuelle. Padam propose en conséquence un produit adapté et clé en main permettant aux transporteurs d'offrir à leurs utilisateurs un service de transport répondant à leurs besoins, économique et facile à utiliser. Les enjeux de

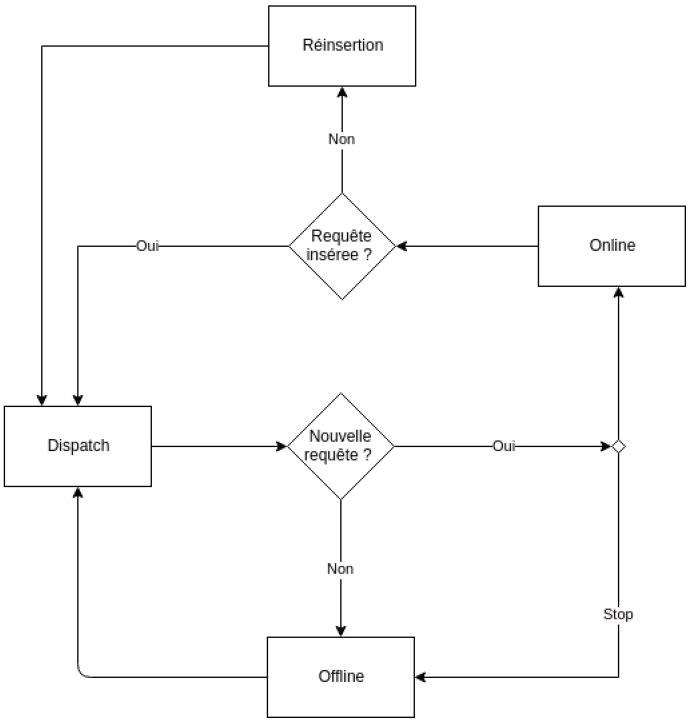


Figure 2.3 – Interaction des différents modules développés dans la thèse. Le module de "Dispatch" (déjà présent dans l'entreprise) gère l'apparition des nouvelles requêtes et orchestre le lancement/arrêt des différents modules

rentabilité et de satisfaction client rencontrés lors de la mise en oeuvre d'un service impliquent naturellement un besoin d'optimisation important. L'objectif de cette thèse est de concevoir et développer des modules d'optimisation efficaces permettant de résoudre un DARP dynamique incluant des contraintes spécifiques au contexte de Padam et permettant de résoudre des instances réelles difficiles. Ces modules répondront à des problématiques peu ou pas traitées dans la littérature telles que l'optimisation sur les points de prise en charge, le choix multiple offert au client, l'optimisation continue des tournées avec flotte finie ou encore la réinsertion temps réel des requêtes rejetées. Avant de présenter nos différentes contributions, le chapitre suivant présente la formalisation du DARP, les spécificités du problème Padam, une synthèse de la littérature sur les problèmes de DARP et les instances utilisées durant la thèse.

Chapitre 3

Description, formulation et état de l'art

3.1 Introduction

Le modèle de la littérature le plus proche du modèle Padam est le DARP (Dial-A-Ride-Problem) ([Cordeau and Laporte, 2007]). Le DARP se décline sous 2 formes : statique et dynamique. Dans la version statique, toutes les requêtes sont connues à l'avance alors qu'elles apparaissent au fur et à mesure dans la version dynamique. Après avoir présenté la formulation classique du DARP en section 3.2.1, nous mettrons en évidence les spécificités du problème de Padam en section 3.2.2. Les sections 3.2.3 et 3.2.4 présenteront un état de l'art des versions statique et dynamique. Nous finirons ce chapitre par la présentation en section 3.3 des instances de la littérature et des instances proposées par Padam sur lesquelles seront réalisées les expérimentations de la thèse.

3.2 DARP

3.2.1 Formulation et présentation générale

Le DARP est une variante des problèmes de *Vehicule Routing Problems* (VRP). Pour un état de l'art récent sur ces problèmes, voir [Toth and Vigo, 2014]. Le facteur humain, qui est la principale différence entre le DARP et la majorité des problèmes de VRP, ajoute une difficulté conséquente en rendant les contraintes horaires plus restrictives.
La formulation classique du DARP s'exprime sous forme d'un programme linéaire en nombre entiers (PLNE) ([Cordeau and Laporte, 2007]). Le réseau routier est représenté par un graphe directionnel $G = (V, E)$ où V est l'ensemble des noeuds et E l'ensemble des arc. L'ensemble des noeuds modélise les points d'arrêt possibles des véhicules et les arcs les trajets entre ces noeuds. A chaque arc $(i, j) \in E$ est associée un poids t_{ij} qui correspond au temps de trajet le plus court entre les noeuds i et j.
L'ensemble des noeuds est partitionné en 3 sous-ensembles : l'ensemble des noeuds de pickup $P = \{1, \ldots, n\}$, l'ensemble des noeuds de dropoff $D = \{n + 1, \ldots, 2n\}$ et 2 copies du noeud de dépôt 0 et $2n + 1$.
Soit K la flotte de véhicule disponible durant le service. La capacité de chaque véhicule est notée Q_k et sa durée de service maximale est de T_k. Chaque demande de transport d'un utilisateur est appelée une requête et est associée un couple $(i, n + i)$ avec $i \in P$ et $n + i \in D$. Chaque requête est caractérisée par la durée requise de service d_i au noeud i, un nombre de passagers associé q_i avec $q_{n+i} = -q_i$. Le temps de trajet maximal pour chaque requête est noté L. Ce temps peut être spécifique à chaque requête i et est dans ce cas noté L_i. Une fenêtre horaire

$[e_i, l_i]$ est associée à chaque noeud i où e_i est l'arrivée au plus tôt à i et l_i est l'arrivée au plus tard à i. Le temps de service au noeud i est noté d_i. La variable principale d'optimisation x_{ij}^k vaut 1 si et seulement si l'arc est traversé par le véhicule $k \in K$. Soient u_i^k l'heure de service au noeud i par le véhicule k, w_i^k le nombre de passagers dans le véhicule k au moment de quitter le noeud i et r_i^k le temps de trajet effectif de l'utilisateur i. Avec ces notations, le modèle peut être formulé ainsi :

$$\text{Minimiser} \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{i,j} x_{i,j}^k \tag{3.1}$$

$$s.c$$

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad \forall i \in P \tag{3.2}$$

$$\sum_{j \in V} x_{i,j}^k = \sum_{j \in V} x_{n+i,j}^k \qquad \forall i \in P, \forall k \in K \tag{3.3}$$

$$\sum_{i \in V} x_{0,i}^k = 1 + \sum_{i \in V} x_{i,2n+1}^k \qquad \forall k \in K \tag{3.4}$$

$$\sum_{j \in V} x_{j,i}^k = \sum_{j \in V} x_{i,j}^k \qquad \forall i \in P \cup D, \forall k \in K \tag{3.5}$$

$$u_j^k \geq (u_i^k + d_i + t_{i,j}) x_{i,j}^k \qquad \forall i, j \in V, \forall k \in K \tag{3.6}$$

$$w_j^k \geq (w_i^k + q_j) x_{i,j}^k \qquad \forall i, j \in V, \forall k \in K \tag{3.7}$$

$$r_i^k \geq u_{n+i}^k - (u_i^k + d_i) \qquad \forall i \in P, \forall k \in K \tag{3.8}$$

$$u_{2n+1}^k - u_0^k \leq T_k \qquad \forall k \in K \tag{3.9}$$

$$e_i \leq u_i^k \leq l_i \qquad \forall i \in P \cup D, \forall k \in K \tag{3.10}$$

$$t_{i,n+i} \leq r_i^k \leq L \qquad \forall i \in P, \forall k \in K \tag{3.11}$$

$$max\{0, q_i\} \leq w_i^k \leq min\{Q_k, Q_k + q_i\} \qquad \forall i \in V, \forall k \in K \tag{3.12}$$

$$x_{i,j}^k \in \{0, 1\} \qquad \forall i, j \in V, \forall k \in K \tag{3.13}$$

L'objectif 3.1 est la minimisation du temps de trajet total des véhicules. 3.2 et 3.3 assurent que chaque réservation n'est traitée qu'une seule fois et que le pickup et le dropoff sont servis par le même véhicule. 3.4 et 3.5 contraignent le véhicule à quitter chaque noeud qu'il visite et à commencer sa tournée et à la finir au dépôt. Les contraintes 3.6 - 3.8 régissent les heures de service, la capacité des véhicules et le temps maximal que l'utilisateur peut passer dans son véhicule. Les contraintes 3.9 - 3.12 assurent que les contraintes de fenêtre horaires et de capacité des véhicules sont respectées.

La majorité des travaux de la littérature sur le DARP considère une flotte homogène de véhicules partant d'un unique dépôt, des fenêtres horaires sur les pickup et dropoff des passagers, un temps de route maximum pour les passagers et des véhicules à capacité limitée. Ces caractéristiques sont parfois traitées en tant que contraintes dures ([Ritzinger et al., 2016], [Chassaing et al., 2016]) ou en tant que contraintes souples dont la violation est pénalisée dans la fonction objectif ([Urra et al., 2015]). Certains travaux considèrent plusieurs dépôts pour les véhicules ([Masmoudi et al., 2016], [Detti et al., 2017]) et une flotte de véhicules et d'utilisateurs hétérogènes transportant par exemple des utilisateurs handicapés ([Ilani et al., 2014], [Lim et al., 2016], [Qu and Bard, 2014]). D'autres intègrent des contraintes de main d'oeuvre

où certaines requêtes doivent être accompagnées par un employé qualifié ([Lim et al., 2016]), où une pause déjeuner doit être programmée pour les chauffeurs ([Zhang et al., 2015]) ou encore où la synchronisation de véhicules et d'assistants doit être assurée ([Liu et al., 2015]). Le transfert de passagers où les clients peuvent être transférés d'un véhicule à un autre durant leur voyage est parfois pris en compte comme dans ([Schönberger, 2017]).

Les objectifs classiques du DARP considérés dans la littérature sont liés à la minimisation des coûts engendrés pour le transporteur (temps de trajet des véhicules, distance parcourue par les véhicules, nombre de véhicules utilisés, temps de travail des chauffeurs...) et la maximisation de la qualité de service (temps de trajet total, déviations par rapport aux heures proposées...) ([Parragh, 2011], [Guerriero et al., 2013], [Coslovich et al., 2006], [Cordeau and Laporte, 2003], [Schilde et al., 2014]). La grande majorité des travaux n'optimise qu'un seul objectif. Parmi les travaux considérant plusieurs objectifs, on distingue 3 types d'approche : transformer les différents objectifs en un seul via une somme pondérée ([Jorgensen et al., 2007], [Kirchler and Calvo, 2013]), utiliser une fonction lexicographique ([Garaix et al., 2010], [Schilde et al., 2014]) ou chercher le front de Pareto du problème ([Parragh et al., 2009], [Núñez et al., 2014]).

Comme nous l'avons indiqué dans la section précédente, le DARP se décline en deux sous-problèmes : statique et dynamique. Pour un état de l'art des travaux avant 2007, voir [Cordeau and Laporte, 2007]. Pour plus de détails sur les travaux des dix dernières années, voir [Ho et al., 2018] qui présente une revue détaillée tant sur la version statique que dynamique. Après avoir présenté les spécificités du DARP de Padam, nous procéderons à une revue de la littérature des travaux récents sur les DARP statique et dynamique.

3.2.2 Spécifications du modèle Padam

Nous modélisons le problème de base rencontré par Padam comme un DARP multi-véhicules statique ([Cordeau and Laporte, 2007]). La modélisation est similaire au modèle exposé section 3.2.1 avec quelques modifications :

- Les véhicules ne commencent pas tous au même dépôt. Un ensemble de noeuds de départs des véhicules est introduit et la contrainte 3.4 est adaptée en conséquence.
- Les véhicules ne sont pas obligés de finir à un noeud précis. Cette contrainte se traduit par l'introduction d'un noeud fictif tel que le temps de trajet de tous les autres noeuds vers ce noeud est nul. Ce noeud est l'équivalent du noeud $2n+1$ dans le modèle classique.
- Les utilisateurs ne peuvent pas attendre dans un véhicule, ce qui signifie que le véhicule doit aller directement au noeud suivant si le véhicule n'est pas vide. Cette contrainte se traduit par l'ajout d'une nouvelle variable binaire s_i^k indiquant si le véhicule est vide en sortie du noeud i. Voici les contraintes associées :

$$w_i^k \leq M\left(1 - s_i^k\right) \qquad \forall i \in P, \forall k \in K \qquad (3.14)$$

$$1 - s_i^k \leq w_i^k \qquad \forall i \in P, \forall k \in K \qquad (3.15)$$

$$u_j^k \leq (u_i^k + d_i + t_{i,j})x_{i,j}^k + Ms_i^k + M(1 - x_{i,j}^k) \qquad \forall i, j \in V, \forall k \in K \qquad (3.16)$$

  Les contraintes 3.14 et 3.15 assurent que la variable s_i^k indique correctement lorsque le véhicule est vide. La contrainte 3.16 empêche l'utilisateur d'attendre dans le véhicule et se linéarise avec la technique du big M.
- Le temps maximal que peut passer un utilisateur dans un véhicule dépend de son temps de trajet. La durée maximale du temps de trajet dans un véhicule pour une requête i

dépend de la durée que prendrait ce trajet en prenant le chemin le plus court entre les noeuds de pickup et dropoff de i multipliée un paramètre fixe γ_i. La valeur de γ_i dépend du temps de trajet direct du client i, ce qui permet de rendre compte du fait que le ressenti d'un client vis à vis des détours est différent suivant la longueur de son trajet direct. La contrainte 3.11 devient donc :

$$t_{i,n+i} \leq r_i^k \leq \gamma_i t_{i,n+i} \quad \forall i \in P, \forall k \in K \tag{3.17}$$

3.2.3 DARP statique

Psaraftis fût le premier en 1980 à formuler et à résoudre le DARP avec un seul véhicule sous forme de programmation dynamique [Psaraftis, 1980]. Dans ce même article, il considéra également le cas dynamique et posa les concepts de base des recherches futures sur les problèmes de VRP dynamique. Une des premières heuristiques pour le DARP statique avec plusieurs véhicules fût présentée en 1986 dans [Jaw et al., 1986]. L'idée de l'heuristique est simple : les requêtes sont triées par heure de pickup croissante, i.e les requêtes ayant une heure de prise en charge plus tôt dans la journée sont rangées en premier. Chaque requête de la liste est ensuite traitée séquentiellement pour déterminer dans quelle tournée elle sera assignée : toutes les insertions possibles de la requête dans chaque tournée sont calculées et celle qui entraîne la plus faible augmentation des coûts est choisie. Si aucune insertion n'est possible, la requête est marquée comme non-assignée et l'opérateur peut si il le désire ajouter une nouvelle tournée pour servir la requête. Cette heuristique et le modèle associé posa les bases de nombreux travaux ultérieurs tant sur le DARP statique que dynamique.

Deux approches principales de résolution ont été proposées depuis : des méthodes exactes comme par exemple les travaux de [Garaix et al., 2010] ou [Garaix et al., 2011] et des méthodes heuristiques comme dans [Hu and Chang, 2015] et [Parragh et al., 2014].

Méthodes exactes

Les méthodes exactes utilisées pour le DARP statique sont pour la plupart basées sur la méthode de branch-and-bound (BB). Une méthode de BB se divise en deux étapes : l'étape de séparation ("branch") et l'étape d'évaluation ("bound").

- La phase de séparation consiste à diviser le problème en un certain nombre de sous-problèmes, en fixant par exemple la valeur d'une des variables $x_{i,j}^k$.
- L'évaluation d'un noeud de l'arbre de recherche a pour but de déterminer l'optimum de l'ensemble des solutions réalisables associé au noeud en question. La résolution de la relaxation continue est en générale réalisée et permet de décider si l'exploration du noeud doit continuer ou si il doit être enlevé de l'arbre.

BB est rarement utilisée seule pour les problèmes d'optimisation industriels complexes. C'est pourquoi d'autres méthodes plus performantes et basées sur BB ont été utilisées pour la résolution du DARP : branch-and-cut (BC), branch-and-price (BP) et branch-and-price-and-cut (BPC). Toutes ces méthodes se concentrent sur des problèmes à un seul objectif.

BC Un algorithme de BC est un BB dans lequel des coupes sont rajoutées aux sous-problèmes de l'arbre du BB. Un des premiers algorithmes de BC pour le DARP fût présenté sur une formulation linéaire à 3 indices dans [Cordeau, 2006] où différentes

familles d'inégalités furent utilisées comme coupes. [Ropke et al., 2007] introduisit ensuite de nouvelles inégalités sur une formulation à deux indices. [Braekers et al., 2014] et [Braekers and Kovacs, 2016] utilisèrent également certaines de ces inégalités. D'autres inégalités seront ensuite trouvées en fonction des caractéristiques du DARP sous-jacent ([Liu et al., 2015], [Braekers and Kovacs, 2016]).

BP Contrairement à BC, les algorithmes de BP se concentrent sur la génération de colonne plutôt que la génération de coupes sur les relaxations continues du BB. BP nécessite de pouvoir formuler le problème d'origine en un problème principal restreint et un sous problème de pricing. Dans le problème restreint, un ensemble de colonnes (variables) est exclu de la relaxation continue pour réduire le temps de calcul. A chaque noeud de l'arbre, des nouvelles colonnes sont générées en résolvant un sous problème de pricing et ajoutées au problème principal dans le but est d'améliorer la relaxation continue. Cette méthode permet de résoudre des programmes linéaires de plus grande taille que BC tout en garantissant la convergence vers l'optimum.

BPC Les algorithmes de BPC sont des algorithmes de BP où des coupes sont ajoutées au fur et à mesure de la procédure. Parmi les travaux récents concernant le DARP, on peut citer [Qu and Bard, 2014] et [Gschwind and Irnich, 2014].

Les plus grandes instances ayant été résolues de manière exacte contiennent 8 véhicules et 96 requêtes. Le temps nécessaire pour résoudre de telles instances est en général de quelques heures au minimum. Ces méthodes semblent ainsi difficilement applicables à des problèmes temps réel et à des instances industrielles réalistes. A ce jour, l'intérêt principal de l'application de méthodes exactes pour le DARP est de pouvoir obtenir des bornes inférieures sur des instances de taille moyenne et des valeurs optimales de référence pour des petites instances. Ces valeurs de référence peuvent servir ensuite à évaluer la performance de méthodes heuristiques. Ce genre de méthodes a en effet été très étudié dans la littérature pour fournir en un temps raisonnable des solutions à des instances de plus grande taille.

Méthodes heuristiques

Devant les limites des méthodes exactes pour résoudre des problèmes à taille réelle, la plupart des travaux menés sur le DARP ont consisté à développer des heuristiques efficaces et rapides. Nombre de ces heuristiques sont basées sur la notion de voisinage et de recherche locale.

— **Voisinage** : Le voisinage d'une solution est un sous-ensemble de solutions qu'il est possible d'atteindre par une série de transformations données. Par extension on désigne parfois par le terme "voisinage" l'ensemble des transformations considérées. Dans le cas d'un problème de DARP par exemple, le voisinage d'une solution (i.e l'ensemble des tournées) peut-être l'ensemble des solutions pouvant être obtenues en insérant une requête d'une tournée vers une autre ou encore en échangeant 2 requêtes de 2 tournées différentes. De nombreux voisinages sont possibles pour un problème donné et c'est le plus souvent des considérations empiriques qui permettent de déterminer les plus efficaces.

— **Recherche locale (LS)** : Une fois un voisinage défini, la recherche locale consiste à passer d'une solution à une autre solution dans le voisinage (le voisinage changeant à chaque nouvelle solution considérée) jusqu'à un critère d'arrêt. Le critère d'arrêt considéré est en général l'obtention d'une solution qui a un coût optimal dans son voisinage et/ou que le temps maximal imparti soit dépassé. Lorsque suffisamment de temps est disponible, la recherche se termine dans un optimum local et ne fournit aucune garantie d'optima-

lité globale dans le cas général. La recherche locale est rarement utilisée seule sur les problèmes de DARP mais plutôt en complément ou en sous-composante d'une autre heuristique.

Nous allons désormais passer en revue les principales heuristiques présentées pour la résolution du DARP statique.

— **Recherche Taboue (TS)** : La recherche taboue est une amélioration de la recherche locale permettant de s'extraire d'un minimum local en acceptant des solutions moins bonnes que la solution courante. Pour éviter d'être à nouveau pris au piège dans le même minimum local, une liste de mouvements tabous est maintenue. Lorsqu'un mouvement vient d'être effectué, il est ajouté à la liste et ne peut pas être effectué durant un nombre d'itérations déterminé.
Cordeau et Laporte furent parmi les premiers à proposer une recherche taboue pour les problèmes de DARP [Cordeau and Laporte, 2003], en incorporant des stratégies de diversification, de pénalisation et en acceptant les solutions non faisables. Etant rapide et performante, elle inspira la plupart des recherches taboues récentes sur des problèmes de DARP avec des contraintes plus compliquées ([Detti et al., 2017], [Kirchler and Calvo, 2013], [Guerriero et al., 2013], [Paquette et al., 2013]). Pour alléger l'étape d'évaluation du voisinage qui est bien souvent la tâche la plus coûteuse, certains travaux proposent de ne considérer les mouvements qu'à l'intérieur d'un certain seuil ([Kirchler and Calvo, 2013]) ou de faire un échantillonage aléatoire ([Detti et al., 2017]). En plus d'être utilisée comme une heuristique à part entière, TS peut aussi être intégrée dans une heuristique de "multi-start" ([Guerriero et al., 2013]).

— **Recherche à voisinage variable (VNS)** : Le principe de base de la VNS est d'utiliser plusieurs structures de voisinage durant la recherche. A chaque itération, chaque voisinage est exploré l'un après l'autre via une recherche locale qui correspond à une phase d'intensification. Un mouvement de perturbation est réalisé après chaque recherche locale pour apporter de la diversification à la recherche.
La première VNS pour le DARP est proposée par [Parragh et al., 2009] en 2009. L'objectif était de résoudre un DARP bi-objectif avec une VNS contenant 4 opérateurs de voisinage dans l'étape de diversification. L'étape de diversification était essentiellement composée par des opérateurs inter-véhicules à l'inverse des opérateurs de recherche locale qui étaient principalement intra-véhicule. Un critère d'acceptation de recuit simulé était utilisé pour sortir la recherche des optima locaux. Les mêmes auteurs proposèrent également une VNS pour le DARP mono-objectif ([Parragh et al., 2010]). Devant les très bons résultats obtenus par la VNS pour les deux problèmes, de nombreuses recherches ont été orientées dans cette direction, parfois sur des variantes plus riches du DARP ([Parragh et al., 2014], [Schilde et al., 2014] [Muelas et al., 2015]).

— **Recherche à voisinage large (LNS / ALNS)** : A chaque itération d'une recherche à voisinage large (LNS), une partie (potentiellement grande) de la solution est détruite puis reconstruite. Ces deux étapes sont effectuées via une heuristique de destruction et de réparation ([Shaw, 1998]). Dans le cas du DARP, la destruction correspond à un nombre k de requêtes qui sont enlevées de leurs tournées respectives. La solution partielle ainsi obtenue après destruction est ensuite reconstruite en y réinsérant les k requêtes enlevées. Une variante célèbre, l'ALNS, fût présentée en 2006 sur un problème de PDVRP dans [Ropke and Pisinger, 2006]. L'idée est d'utiliser plusieurs heuristiques de destruction et de réparation durant la recherche. A chaque itération, le choix de l'heuristique de destruction et de réparation dépend des performances de chaque heuristique sur les itérations précédentes. Le critère d'acceptation d'une nouvelle solution est

de type recuit simulé. L'heuristique proposée est très efficace et a posé les fondations d'études plus récentes de LNS sur le DARP comme [Häll and Peterson, 2013], [Masmoudi et al., 2016], [Masson et al., 2014] ou [Lehuédé et al., 2014]. En plus d'être une méthode d'optimisation à part entière, l'ALNS peut être incorporée dans un système multi-critère ([Lehuédé et al., 2014]) et dans une heuristique à plusieurs redémarrages ([Qu and Bard, 2013]). [Gschwind and Drexl, 2019] reprit l'ALNS de [Ropke and Pisinger, 2006] et ajouta trois opérateurs de destruction proposés par [Masson et al., 2013] et [Parragh et al., 2010]. La version ainsi proposée de l'ALNS obtint les meilleures résultats sur les instances du DARP derrière l'algorithme génétique de [Masmoudi et al., 2017]. En rajoutant une amélioration par le voisinage de Balas-Simonetti ([Balas and Simonetti, 2001]) après chaque réparation et la résolution en fin d'ALNS d'un problème de recouvrement, l'algorithme obtint les meilleurs résultats, devant [Masmoudi et al., 2017].

— **Algorithmes génétiques (GA)** : Les algorithmes génétiques sont des algorithmes à base de population. La recherche commence avec une population initiale d'individus (solutions). A chaque itération, les individus sont choisis pour être les parents de futurs individus. Ce choix est probabiliste et s'effectue suivant les valeurs données par une fonction d'évaluation. De nouveaux individus sont créés en appliquant des opérateurs de mutation et de croisement sur les parents. Certains parents pourront ensuite être remplacés par des individus fils lors de la prochaine étape de sélection. La recherche se poursuit jusqu'à atteindre un critère d'arrêt comme le nombre de générations ou lorsque la qualité de la solution est suffisante. Les algorithmes génétiques sont utilisés depuis longtemps dans les problèmes de VRP ([Baker and Ayechew, 2003], [Rachid, 2010]). [Jorgensen et al., 2007] et [Cubillos et al., 2009] ont présenté une approche de "cluster-first, route second" avec une alternance entre GA pour la construction des clusters et une heuristique gloutonne pour la construction des routes. On peut noter également [Atahran et al., 2014] qui incorpore une GA dans une approche multi-objectif. La majorité des travaux récents en rapport avec GA se sont néanmoins concentrés sur des approches GA hybrides.

— **Approches métaheuristiques hybrides** : Il existe deux manières principales d'hybrider une métaheuristique : (1) chaque métaheuristique est exécutée de manière séquentielle ou (2) une métaheuristique est exécutée à l'intérieur d'une autre. [Parragh et al., 2009], [Santos and Xavier, 2015] et [Molenbruch et al., 2017] sont des exemples de la première procédure où des techniques de "path relinking" sont respectivement appliquées après une VNS, GRASP et une LS multi-directionelle. Les meilleures solutions sont récupérées des recherches respectives, l'idée étant d'obtenir de meilleures solutions en explorant les trajectoires entre ces dernières. La manière la plus habituelle d'exploiter la méthode (2) est d'intégrer une métaheuristique à voisinage comme une recherche locale ou une LNS dans une métaheuristique à population telle que GA. L'intérêt de ce genre de méthodes est de combiner la capacité d'exploration des algorithmes à population avec la capacité d'intensification des méthodes à voisinage. [Chassaing et al., 2016], [Chevrier et al., 2012], [Zhang et al., 2015], [Masmoudi et al., 2016], [Masmoudi et al., 2017] et [Schönberger, 2017] sont des exemples de travaux utilisant cette approche, GA étant la métaheuristique de population la plus utilisée et LS la métaheuristique de voisinage la plus utilisée. [Masmoudi et al., 2017] présenta en 2017 une méthode hybride combinant GA et LS. Cette méthode est actuellement la meilleure sur les problèmes de DARP hétérogènes. D'autres approches hybrides ont été tentées, en remplaçant l'étape de recherche locale par une méthode de LNS dans GRASP ([Pimenta et al., 2017]) ou avec une descente à voisinage variable dans une ILS (Iterated Local Search) ([Lim et al., 2016]).

Actuellement, l'ALNS hybride de [Gschwind and Drexl, 2019] est la meilleure heuristique sur les instances du DARP, suivi par la GA hybride de [Masmoudi et al., 2017] et l'ALNS simple de [Gschwind and Drexl, 2019]. La GA hybride est néanmoins la plus efficace sur les instances hétérogènes du DARP tandis que l'algorithme de colonie d'abeille de [Masmoudi et al., 2016] obtient les meilleurs résultats sur le DARP hétérogène avec plusieurs dépôts.

3.2.4 DARP Dynamique

Lorsqu'une partie des requêtes n'est pas connue au moment de l'optimisation, on parle de DARP dynamique. Le DARP dynamique se rencontre fréquemment lorsqu'une partie des requêtes arrive en temps réel durant le déroulement du service et/ou lorsqu'une réponse immédiate est attendue au moment où le client réserve un trajet. Cette particularité nécessite le développement de techniques capables de gérer l'afflux de requêtes tout en produisant des solutions de bonne qualité. Ainsi, de nombreux travaux sur le DARP dynamique se sont concentrés sur des heuristiques rapides pour insérer les nouvelles requêtes (inspirées par [Jaw et al., 1986], décrite en section 3.2.3) et/ou des méthodes métaheuristiques pour apporter une optimisation plus intense aux tournées construites ([Attanasio et al., 2004], [Coslovich et al., 2006], [Santos and Xavier, 2015]). Certains travaux cherchent à utiliser des informations stochastiques, comme l'apparition des futures requêtes ou la prédiction du traffic par exemple ([Schilde et al., 2011], [Schilde et al., 2014], [Sayarshad and Chow, 2015], [Núñez et al., 2014]). On parle alors de DARP Dynamique et Stochastique. D'autres approches se concentrent sur l'étude des paramètres de service importants dans un DARP dynamique ([Häll et al., 2012], [Häll et al., 2015]). Les différentes méthodes de résolution rencontrées se répartissent en 4 groupes :

1. Heuristiques d'insertion pour une réponse temps réel aux clients
2. Heuristiques d'insertion suivi d'une phase d'optimisation entre deux évènements (apparition d'une requête, arrivée d'un véhicule à une place...)
3. Heuristiques optimisant en continu et intégrant (régulièrement ou immédiatement) les nouvelles requêtes
4. Heuristiques d'insertion suivie d'heuristiques de réinsertion pour les requêtes rejetées

Heuristiques d'insertion

[Maalouf et al., 2014] étudie un cas de DARP où les clients choisissent uniquement une heure maximal et minimal de dropoff, l'heure de pickup étant déterminée par la durée maximum autorisé pour chaque client. Une heuristique d'insertion simple et rapide basée sur des concepts de logique floue est présentée, avec le double objectif de minimiser le temps passé par un client dans un véhicule et de minimiser la distance parcourue par le véhicule associé. L'heuristique est testée sur deux instances générées aléatoirement avec 6 véhicules et 100 requêtes et 30 véhicules et 900 requêtes. L'heuristique permet d'insérer une nouvelle requête en moins de 0.25s et peut ainsi être utilisée dans un service de transport en temps réel.

[Wong et al., 2014] étudie l'efficacité d'un système de DARP en fonction de la proportion de requêtes temps réel. Les requêtes en avance sont insérées séquentiellement avec l'heuristique d'insertion proposée par [Jaw et al., 1986] suivie d'une phase d'amélioration via des échanges de requête à leur meilleure position. Lorsqu'une requête temps réel arrive, la même heuristique d'insertion est utilisée pour choisir le véhicule dans lequel le nouveau client sera inséré. Si aucun véhicule n'est disponible et que le nombre maximal de véhicules n'est pas encore atteint, un nouveau véhicule est rajouté pour servir le nouveau client. Un algorithme d'ordonnancement

calcule ensuite les heures d'arrivée et de départ des différents arrêts du véhicule où le client a été inséré. Plusieurs stratégies d'ordonnancement sont proposées et testées. Les expérimentations ont lieu sur des groupes de 30 instances générées aléatoirement, chaque groupe comportant respectivement 100, 500 et 1000 requêtes. Chaque instance a un pourcentage de requêtes dynamique noté DOD. L'objectif est de minimiser le nombre de véhicules utilisés et le nombre total de kilomètres parcourus par les véhicules utilisés. Les résultats montrent que le coût de transport n'est pas linéaire en fonction du DOD et qu'il atteint son pire niveau autour de 70%. Ils permettent également aux opérateurs d'adapter certains éléments de leur politique en fonction du dynamisme des instances.

Heuristiques d'insertion et post-optimisation

Dans [Coslovich et al., 2006], une partie des clients arrive de manière imprévue. Le problème d'optimisation est traité indépendemment pour chaque véhicule avec l'objectif de maximiser la qualité de service des clients déjà insérés. Une recherche locale s'exécute en permanence entre les arrêts consécutifs du véhicule pour optimiser les tournées pendant le temps disponible. Lorsqu'une nouvelle requête arrive à un arrêt du véhicule, une heuristique d'insertion rapide est lancée sur le voisinage de la solution actuelle pour répondre rapidement au client. Les expérimentations sont effectuées sur 540 instances générées par les auteurs et montrent que les performances obtenues par la procédure présentée sont proches de la solution obtenue en considérant toutes les requêtes comme étant connues à l'avance.

Dans [Beaudry et al., 2010], les auteurs modélisent un système de transport hospitalier allemand avec des contraintes spécifiques au contexte médical sous-jacent. L'objectif est de minimiser une somme pondérée du temps de trajet des véhicules, de l'avance et du retard par rapport aux heures de service des clients. Les requêtes des utilisateurs ont ainsi des fenêtres horaires souples dont la violation est pénalisée dans la fonction objectif. Une heuristique d'insertion est lancée dès lors qu'une requête se présente, suivie par une phase d'amélioration basée sur une recherche taboue similaire à [Cordeau and Laporte, 2003] et ayant lieu entre l'apparition des requêtes. La procédure présentée est testée sur des instances basées sur 20 jours de données réelles de l'hôpital ayant motivé l'étude. Les expérimentations montrent que la phase d'amélioration apporte un gain significatif sur la procédure d'insertion et que le temps d'attente des patients est réduit tout en utilisant moins de véhicules.

Dans [Häll et al., 2012], les auteurs présentent un outil de modélisation pour la simulation de DARP dynamique. Ce dernier permet de traiter les requêtes en temps réel avec l'heuristique de [Jaw et al., 1986] et d'optimiser les tournées en utilisant une heuristique de destruction/réparation. L'heuristique de ré-optimisation présentée dans l'article consiste en une simple alternance de retrait/réinsertion des requêtes les unes après les autres. La fonction objectif est une combinaison pondérée du coût induit par le temps de trajet total des véhicules et de la qualité de service du client. L'outil présenté est illustré sur une instance de 3072 requêtes représentant une journée d'un service réel à Göteborg en Suède. Le nombre de bus est considéré comme étant infini. Dans [Häll and Peterson, 2013], les auteurs utilisent l'outil de simulation pour étudier l'impact de divers opérateurs de construction/ reconstruction dans la phase de réoptimisation et montrent l'intérêt de telles procédures dans un système de DARP dynamique en obtenant une réduction des coûts jusqu'à 5% sur une instance de 606 requêtes provenant d'un service réel de la ville de Göteborg. Toujours avec ce même outil de simulation, [Häll et al., 2015] étudie les réponses du système en fonction de certains paramètres influant la qualité de service des clients (largeur des fenêtres, temps maximal dans le véhicule...).

[Marković et al., 2015] propose un système d'optimisation appelé MRMS appliqué à un problème réel issu de l'état du Maryland où environ 90% des requêtes sont connues en avance. La méthodologie proposée est principalement concentrée sur la résolution du DARP statique, pour lequel ils utilisent l'heuristique d'insertion parallèle de [Jaw et al., 1986] en rajoutant deux opérateurs simples d'amélioration une fois toutes les requêtes insérées. Leur solution est dans un premier temps testée et comparée à la solution optimale sur les instances de [Cordeau, 2006]. MRMS est ensuite utilisée sur le cas réel d'une compagnie de taxis du Maryland, avec environ 450 requêtes sur une journée. $MRMS$ utilise 34 véhicules pour servir les requêtes et permet d'obtenir une réduction de 18% des coûts en comparaison avec une solution obtenue manuellement par les opérateurs. Les auteurs proposent ensuite une manière d'adapter MRMS aux requêtes temps réel en utilisant l'heuristique d'insertion pour une réponse immédiate puis les opérateurs d'amélioration pour une optimisation continue. Faute de pouvoir comparer avec une solution manuelle, ils ne présentent aucune simulation utilisant MRMS dans un cadre dynamique.

Optimisation en continu

[Attanasio et al., 2004] utilise la recherche taboue parallèle proposée par [Cordeau and Laporte, 2003] dans un cadre dynamique. La fonction objectif est le coût total du déplacement des véhicules auquel s'ajoute une combinaison pondérée de la violation des différentes contraintes temporelles. Une solution initiale est construite grâce à la recherche taboue avec l'ensemble des requêtes statiques. Durant le déroulement du service, l'algorithme tourne entre l'apparition de deux requêtes successives. Lorsqu'une nouvelle requête arrive, une étape de vérification de la faisabilité limitée à 30s est enclenchée pour tenter d'insérer la nouvelle requête dans les tournées : une recherche taboue incorporant la nouvelle requête est lancée en parallèle sur différents fils d'exécution (threads) indépendants pour trouver au plus vite une solution faisable. Les auteurs testent 2 stratégies de parallélisme pour la TS, 2 stratégies pour la construction de la solution statique et 2 pénalisations différentes des contraintes pour l'étape de faisabilité. Les expérimentations sont faites sur les 26 instances statiques proposées par [Cordeau and Laporte, 2003] (en choisissant aléatoirement une moitié des requêtes pour être dynamiques) et présentent le coût final de la solution et le pourcentage de requêtes servies. Les résultats montrent que le fait d'augmenter le nombre de threads permet d'augmenter le nombre de requêtes servies par le système avec dans la plupart des cas une augmentation du coût des tournées associées.

[Schilde et al., 2011] travaille sur une version stochastique du DARP dynamique en utilisant les informations sur les trajets retour des clients pour améliorer la qualité de la solution. La fonction objectif est lexicographique avec comme premier objectif la minimisation du retard total, puis le nombre de véhicules et enfin la durée de trajet totale. Chaque requête est associée à une fenêtre horaire souple (pénalisée par la fonction objectif) de telle sorte qu'aucune requête n'est jamais rejetée. Le système vérifie régulièrement si de nouvelles requêtes sont apparues et les incorpore dans la procédure d'optimisation. Les auteurs adaptent une VNS en proposant une version dynamique et une version stochastique puis conduisent une étude comparative en incluant une MPA (Multiple Plan Approach) et sa version stochastique MSA (Multiple Scenario Approach) ([Bent and Van Hentenryck, 2004]). L'idée de MPA est de maintenir un ensemble de solutions (plans) compatibles avec les informations actuelles à la disposition du système. Une fonction de rang est définie pour indiquer la solution qui servira de base lorsque des décisions doivent être prises. MSA reprend un fonctionnement similaire en ajoutant les informations sur les futures requêtes via une distribution d'échantillonage. Pour tester les différents algo-

rithmes, des instances sont générées à partir d'une année entière de données concernant la ville de Graz en Autriche. Les résultats montrent que l'ajout d'informations stochastiques apporte un bénéfice lorsque le nombre de trajets retour est faible en comparaison avec le nombre total de requêtes et lorsque ces informations ne sont pas trop loin dans le temps. De plus, la version dynamique de VNS semble donner des résultats meilleurs que MSA.

Les mêmes auteurs reprennent les mêmes métaheuristiques dans [Schilde et al., 2014] dans le but d'évaluer le bénéfice apporté par l'ajout d'informations stochastiques sur les temps de trajet entre les noeuds du réseau routier. Les instances utilisées sont générées à partir des données historiques de 2009 sur la ville de Vienne. Les résultats montrent que MSA n'apporte pas de plus-value pour ce problème et que la VNS dynamique apporte une amélioration significative lorsque le pourcentage de requêtes inconnues se situe entre 45 et 90%.

[Santos and Xavier, 2015] présente un problème de taxi et partage de trajets dynamique en prenant en compte les contraintes de prix. La journée est divisée en intervalles de temps de même taille. Durant chaque intervalle, une instance du DARP statique est résolue avec l'ensemble actuel des requêtes via une heuristique de GRASP ([Feo and Resende, 1995]). Les auteurs utilisent leur heuristique à la fois pour les cas statique et dynamique. Pour le cas statique, 80 instances basées sur des données de Sao Paulo sont présentées. Les expérimentations montrent l'efficacité de l'heuristique proposée en comparaison avec leur précédente heuristique ([Santos and Xavier, 2013]). Concernant la version dynamique, 4 instances basées sur le réseau routier de Sao Paulo sont présentées. L'ensemble des véhicules disponible est dynamique : chaque heure, 1333 nouveaux véhicules sont disponibles pour une durée de 4h. 54 requêtes aléatoires sont générées chaque minute et plusieurs discrétisations de temps sont testées (2min 30, 5 min et 10 min). Les résultats montrent que 5 minutes est la valeur la plus adéquate et que l'heuristique présentée permet en moyenne d'apporter jusqu'à 30% de gain économique pour les clients.

Dans [Lois and Ziliaskopoulos, 2017], les auteurs présentent et combinent deux modules d'optimisation pour résoudre un problème de DARP dynamique. Le premier contient une heuristique d'insertion gloutonne destinée à répondre aux nouvelles requêtes. Le deuxième, REGRET, est destiné à optimiser les tournées pendant le temps libre du service. L'heuristique de regret qui le compose calcule pour chaque requête une valeur de regret, évaluant le profit obtenu à déplacer cette requête de son véhicule actuel vers un autre. La requête avec le plus grand regret est choisie et la procédure est itérée jusqu'à ce que plus aucune amélioration ne soit possible. L'ensemble des itérations réalisées jusqu'à ne plus obtenir d'amélioration est appelé un cycle. REGRET tourne en continue pendant le service et après chaque cycle, le système vérifie si une nouvelle requête est apparue. Si c'est le cas, elle est traitée par l'heuristique d'insertion sinon REGRET recommence un nouveau cycle. 9 instances basées sur des données de Philippi comportant de 94 à 1619 requêtes et jusqu'à 50 véhicules sont utilisées. Le temps maximal qu'un client est prêt à attendre avant d'obtenir une réponse par l'algorithme d'insertion est de 60s. Les résultats montrent que l'heuristique d'insertion est capable d'insérer toutes les requêtes pour chaque instance. Lorsque le module REGRET est ajouté, un profit jusqu'à 4.3% est gagné sur les 4 instances inférieures à 700 clients tandis que le profit diminue jusqu'à -40% pour les instances plus grandes. Ceci est en grande partie due à la durée d'un cycle d'optimisation, qui peut durer jusqu'à 20 minutes et ainsi refuser de nombreux clients se présentant dans l'intervalle.

Heuristiques d'insertion et Réinsertion

Nous n'avons trouvé qu'un seul article utilisant cette méthodologie consistant à réinsérer une requête refusée au préalable par le système : il s'agît de [Luo and Schonfeld, 2011] dans lequel les auteurs adaptent une heuristique d'insertion parallèle et une heuristique de réinsertion proposées pour le DARP statique ([Luo and Schonfeld, 2007]). Chaque fois que l'heuristique d'insertion rejette un client, une heuristique de réinsertion est utilisée pour tenter de l'insérer en déplaçant d'autres clients considérés comme similaires. Deux stratégies temps réel sont proposées : "insertion immédiate" dans laquelle une requête est insérée dès qu'elle apparaît et "insertion à horizon variable" où l'insertion d'une requête avec une heure de service demandée éloignée de l'instant présent est reportée à plus tard. L'objectif global est de réduire le nombre de véhicules utilisés. Les auteurs utilisent les instances qu'ils ont précédemment proposées dans [Luo and Schonfeld, 2007] pour le DARP statique en les adaptant à leur contexte dynamique. Les résultats montrent la supériorité de la politique à horizon variable en permettant une réduction du nombre de véhicules jusqu'à 10% par rapport à l'heuristique parallèle d'insertion.

3.3 Présentation des instances

De nombreux travaux ont été menés sur le DARP statique et ses différentes variantes. Les 10 dernières années ont ainsi vu l'émergence de nombreuses méthodes de résolution différentes, tant exactes que heuristiques. Ces diverses techniques ont inspiré la plupart des travaux sur le DARP dynamique, qui nécessite un traitement particulier du fait de son aspect temps réel. Cependant, beaucoup moins de travaux ont été menés sur ce dernier en comparaison avec la version statique. De plus, aucun des problèmes rencontrés ne rassemble toutes les particularités du système de transport de Padam. Certains ne considèrent qu'un seul véhicule ([Coslovich et al., 2006]), utilisent un nombre infini de véhicules ([Wong et al., 2014], [Häll et al., 2012]) ou considèrent des fenêtres temporelles souples pour les clients ([Beaudry et al., 2010]). L'impact d'une procédure de post-optimisation sur le nombre total de requêtes servies avec une flotte finie n'a pas été clairement étudié. Les travaux utilisant des méthodes d'optimisation en continu acceptent des temps de réponse client trop longs ([Attanasio et al., 2004], [Lois and Ziliaskopoulos, 2017]) ou ne traitent pas les clients immédiatement lorsque ces derniers soumettent leur requête ([Santos and Xavier, 2015]). De plus, le sujet de la réinsertion n'a été (à notre connaissance) traité que dans [Luo and Schonfeld, 2011]. Cependant, la flotte considérée est infinie et l'heuristique proposée est sujette à de nombreuses améliorations. D'autres défis important soulevés par la problématique de Padam ne sont pas non plus représentés dans la littérature. Ainsi, tous les travaux rencontrés proposent une solution unique au client en laissant de côté la complexité liée à la possibilité de choix multiple qui est fondamentale pour l'entreprise. De plus, le point de pickup/dropoff étant toujours celui du client ou le plus proche de son adresse, aucun travail sur l'optimisation des points de ramassage et dépose n'a été mené.

Plusieurs ensembles d'instances ont en revanche été proposés pour permettre aux chercheurs de tester et comparer leurs méthodologies pour le DARP statique. Concernant le DARP dynamique, la plupart des travaux sont orientés sur des problèmes industriels spécifiques et utilisent des instances associées au contexte étudié. Il n'existe donc pas d'instances standards pour le DARP dynamique. Nous utiliserons donc des instances fournies par Padam pour tester les algorithmes proposés.

3.3.1 Instances DARP statique

Pour tester les différentes méthodes proposées, plusieurs ensembles d'instances générées artificiellement et basées sur des données réelles ont été proposés. Beaucoup sont accessibles publiquement sur Internet.

- 26 instances ont été proposées dans [Cordeau and Laporte, 2003]. 20 de ces instances sont générées aléatoirement en prenant en compte certaines informations fournies par la Commission du Transit de Montréal (MTC). Elles contiennent entre 3 et 13 véhicules et entre 24 et 144 requêtes. Les 6 instances restantes sont des données réelles d'un service de transport Danois et contiennent entre 200 et 295 requêtes. Les 26 instances sont téléchargeables à l'url `http://www.crt.umontreal.ca/~cordeau/data`.
- 30 autres instances de plus petite taille furent proposées dans [Cordeau, 2006] pour tester une méthode de résolution exacte. Les instances sont présentées en 2 groupes de 15 et contiennent de 2 à 4 véhicules et de 16 à 32 requêtes. La différence entre les deux groupes réside dans la taille des véhicules (3 et 6) et le temps maximum qu'un client peut passer dans un véhicule (30 et 60 minutes). Ces instances sont téléchargeables à l'url `http://neumann.hec.ca/chairedistributique/data/darp/branch-and-cut/`.
- Les instances de [Cordeau, 2006] ont été réutilisées et étendues dans [Ropke et al., 2007] où les auteurs présentent un ensemble similaire de 42 instances (comprenant en partie celles de [Cordeau, 2006]) de 2 à 8 véhicules et de 16 à 96 requêtes. Ces instances sont disponibles à l'url `http://www.hec.ca/chairedistributique/data` et constituent les plus grandes instances du DARP à avoir été résolues de manière exacte.
- D'autres instances ont également été proposées pour le DARP hétérogène et le DARP hétérogène multi-dépôts dans [Parragh, 2011] et [Braekers et al., 2014] respectivement.

3.3.2 Instances de Padam

Toutes les expérimentations effectuées dans le cadre de cette thèse sont menées sur des instances réelles fournies par Padam. Nous présentons ici leurs caractéristiques principales sur les plans géographiques et de la qualité de service fournie.

Nous utilisons 19 instances divisées en deux groupes, que nous appelons A et B. Le groupe A contient 10 instances et le groupe B en contient 9. Chaque groupe modélise un système de transport différent avec ses propres caractéristiques géographiques et de service. Le groupe A correspond au cas d'étude de *Slide* dans la ville de Bristol tandis que le cas B correspond à un cas d'étude dans le 19^{eme} arrondissement de la ville de Paris. Chaque instance est appelée G_N, où G est le nom du groupe et N le nombre de requêtes. Le tableau 3.1 présente les informations concernant la flotte de véhicules et la géographie sous-jacente. La colonne *Noeuds* indique le nombre de noeuds du graphe sous-jacent (noeuds de pickup/dropoff et de parking), *Durée du service* la durée du service, *Capacité* la capacité des véhicules, *Surface* la surface en km^2 couverte par le graphe et *Flotte* le nombre de véhicules par défaut décidé entre *Padam* et l'agence de transport partenaire. Dans certaines simulations, nous ferons varier ce nombre pour évaluer le comportement de nos algorithmes en fonction de la taille de la flotte ou essayer d'obtenir des résultats similaires avec moins de véhicules. Enfin, la colonne *Requêtes* donne le minimum/maximum de requêtes parmi les instances du groupe et la colonne *Avance* le pourcentage moyen de requêtes effectuées avant le début du service (en avance).

On constate d'après le tableau 3.1 que les deux groupes d'instances ont des caractéristiques très différentes : A est un grand territoire avec un service de courte durée représentant un

Groupe	Noeuds	Durée du service	Capacité	Surface (km^2)	Flotte	Requêtes	Avance
A	473	3h30	8	25	13	120 - 360	66%
B	90	12h	30	5	6	200 - 1011	36%

Tableau 3.1 – Principales caractéristiques des groupes A et B

service de "commuting" (matin ou soir) avec des mini-bus tandis que B correspond à un territoire beaucoup plus petit représentant un service de transport public dans un quartier en zone urbaine.

Le tableau 3.2 présente les contraintes clients associées à chaque groupe, qui représente la qualité de service offerte aux utilisateurs. Les colonnes PWB/PWA et DWB/DWA définissent la taille des fenêtres horaires des pickup et dropoff (respectivement) : la largeur de la fenêtre horaire autour d'un noeud de pickup (resp. dropoff) d'un client quelconque est égale à $PWA + PWB$ (resp. $DWA + DWB$). L'utilisation précise de ces paramètres est expliquée en section 4.2.4. *Valeurs gamma* présente les différentes valeurs des gammas et *Seuils gamma* les intervalles de temps dans lesquels ces valeurs s'appliquent (cf équation 3.17). W définit la déviation maximale qu'un client est prêt à accepter par rapport à l'heure demandée (cf chapitre 4) et TPC le temps consacré à la dépose/prise en charge des clients à une place du véhicule. Ce temps est toujours le même, quelque soit le nombre de clients montant/descendant à une place. Tous ces paramètres sont exprimés en minutes. Les valeurs du tableau 3.2 indiquent que les deux groupes d'instances ont une qualité de service relativement similaire. La principale différence concerne les gammas, qui ont des seuils plus petits dans le groupe B pour s'adapter à des temps de trajet plus courts. Les deux groupes présentent des fenêtres horaire restreintes (10 minutes dans certains cas) qui assurent une haute qualité de service aux clients mais rendent les instances difficiles. Basée sur la connaissance des instances et sur l'expérience opérationnelle de Padam, nous pouvons affirmer que les instances du groupe A sont plus difficiles que celles du groupe B. En effet, le territoire associé au groupe A est beaucoup plus grand et les requêtes des clients ne suivent pas des pattern géographiques aussi réguliers que les requêtes du groupe B, ce qui rend plus difficile la mutualisation des utilisateurs.

Groupe	PWB/PWA	DWB/DWA	Seuils gamma	Valeurs gamma	W	TPC
A	0/10	10/13	$]0, 10]$ / $]10, 20]$ / $]20, \infty]$	2 / 1.8 / 1.7	20	1
B	5/8	10/10	$]0, 5]$ / $]5, 10]$ / $]10, \infty]$	2.5 / 2 / 1.8	20	0.5

Tableau 3.2 – Qualité de service pour chaque groupe d'instance. Les paramètres temporels sont exprimés en minutes

Les deux groupes d'instances présentent ainsi des contraintes de qualité de service qui les rendent difficiles. De plus, les différences géographiques et de demande nous permettront d'évaluer le comportement des algorithmes présentés dans des contextes de transport réalistes variés.

3.4 Conclusion

Dans ce chapitre, nous avons présenté une synthèse des différents travaux sur les versions statique et dynamique du DARP. Nous avons vu que, contrairement à la version statique, la ver-

sion dynamique a été peu étudiée. Après avoir présenté les spécificités du problème de Padam, nous avons pu mettre en évidence un manque dans la littérature et la nécessité de développer des méthodes spécifiques à notre contexte industriel. Nous avons de plus présenté les instances fournies par Padam et sur lesquelles les expérimentations de la thèse vont être menées.

Dans le chapitre suivant, nous présentons la première contribution de la thèse : le module online, dont le but est de répondre en temps réel aux utilisateurs.

Chapitre 4

Module Online

4.1 Introduction

Dans ce chapitre, nous présentons en détails le module online, qui est la première contribution de la thèse. Le module est lancé à chaque fois qu'un utilisateur cherche à réserver en avance ou en temps réel. Son objectif principal est de construire les tournées des différents véhicules au fur et à mesure de l'arrivée des requêtes des utilisateurs. Après avoir donné une vue d'ensemble de la problématique en section 4.2, nous présenterons l'heuristique multi-horaires au coeur du module en section 4.3. Des expérimentations sur les instances présentées en section 3.3.2 seront menées dans la section 4.5. Nous terminerons par une conclusion sur les expérimentations.

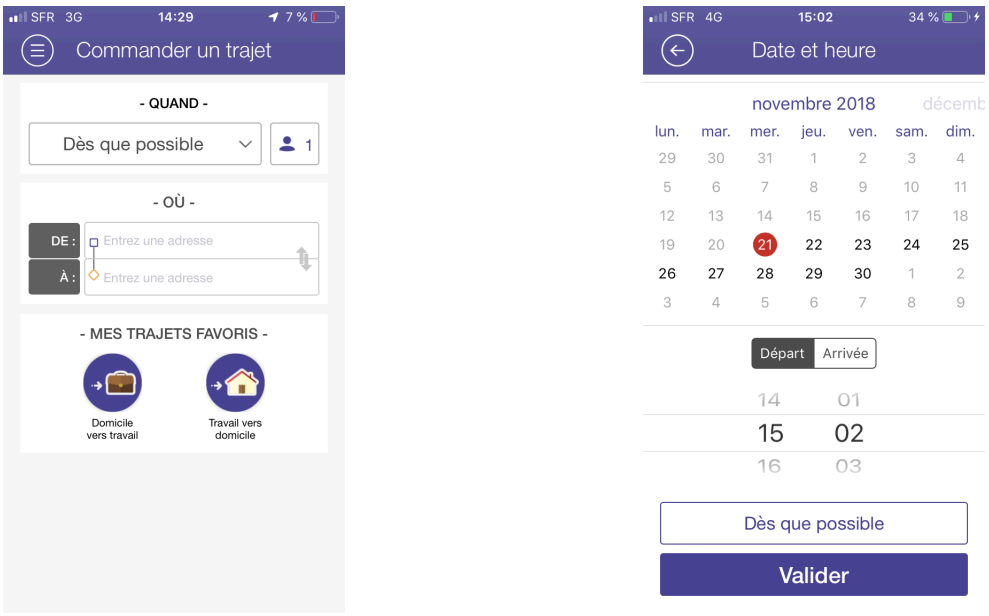
4.2 Présentation de la problématique

Lorsque le service est ouvert à la réservation (un à plusieurs jours avant son début effectif), les tournées des véhicules sont vides. Pour utiliser le service, l'utilisateur dispose de 2 applications mobiles sur les plate-formes IOS et Android ainsi que d'un site internet pour réserver. Une fois inscrit sur l'une d'elles, il peut utiliser librement l'application ou le site pour réserver des trajets. Il peut réserver en avance, c'est-à-dire avant le début du service, ou en temps réel une fois que le service a commencé et que les véhicules ont commencé à rouler. Le processus de réservation est exactement le même dans les 2 cas. Les requêtes des utilisateurs sont transmises au module online qui se charge de la création progressive des tournées. Les sous-sections suivantes présentent une vue d'ensemble de chacune des étapes du module.

4.2.1 Données d'entrée

La figure 4.1 présente 2 captures écran du processus de réservation de l'application IOS. Comme indiqué sur l'image, l'utilisateur doit spécifier :

— L'heure de service désirée notée RH
— Si RH concerne le pickup ou le dropoff (cases "Départ" ou "Arrivée" de 4.1b). Le client est dit PO (resp. DO) si l'heure concerne le pickup (resp. dropoff)
— Le nombre de personnes concernées par la réservation
— L'adresse de départ
— L'adresse d'arrivée



(a) Page principale

(b) Page spécification horaire

Figure 4.1 – Capture écran des pages associées à la réservation sur l'application IOS. L'image 4.1b correspond à la page affichée lorsque l'utilisateur appuie sur la case "Dès que possible" de l'image 4.1a.

Ces informations forment l'ensemble des données d'entrées du module concernant la requête de l'utilisateur.

4.2.2 Contraintes et défis

La figure 4.2 montre une capture écran de la page de réponse du module online au client.

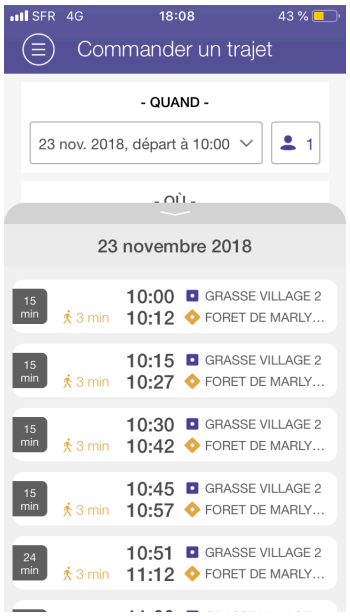


Figure 4.2 – Capture écran de la page de réponse. Le client peut faire défiler pour voir les propositions suivantes.

La réponse attendue par le module online est donc un ensemble de propositions réparties dans le temps où chaque proposition indique :

— Les noeuds du maillage (cf section 3.3.2) auxquels l'utilisateur sera pris en charge et déposé
— Le temps de marche total entre ces noeuds et les adresses renseignées
— Les heures de prise en charge et de dépose aux noeuds de pickup et dropoff

Pour déterminer les propositions, il dispose d'un nombre fixe de tournées dans lesquelles il doit insérer la requête de l'utilisateur tout en respectant les contraintes horaires serrées (cf section 3.3.2) des utilisateurs déjà insérés. Malgré ces contraintes, le module online doit être capable de fournir une réponse en quelques secondes tout au plus.

4.2.3 Objectifs opérationnels et scientifiques

Le module online constitue pour Padam le moyen d'interaction principal avec le client. Il est donc très important de développer un module online capable de créer des tournées de haute qualité en respectant toutes les contraintes imposées par le système. A notre connaissance, il n'existe pas de travaux dans la littérature permettant de traiter l'aspect multi-proposition propre au système de transport en commun opéré par Padam. De plus, aucun des travaux rencontrés n'intègre les points de pickup et dropoff des utilisateurs dans le processus d'optimisation (cf section 3.2.4). L'enjeu du module online est donc à la fois opérationnel et scientifique.

4.2.4 Processus de validation

Après avoir reçu un ensemble de propositions, l'utilisateur est libre de choisir ou non l'une d'entre elles. Dans le cas où il accepte l'une d'elles, il se peut que son choix ait lieu plusieurs dizaines de secondes voir plusieurs minutes après l'instant où les propositions lui ont été faites. Il est donc possible que d'autres personnes aient réservé entre temps et que la proposition choisie ne soit plus faisable. Le module online assure que la proposition faite à l'utilisateur est toujours disponible. Pour s'en assurer, une étape de validation invisible pour ce dernier est réalisée. Si la proposition est toujours disponible, l'utilisateur est assigné à une tournée et dirigé vers la page présentée sur l'image 4.3. Dans le cas contraire, un message est affiché à l'utilisateur qui est automatiquement redirigé vers l'écran de réservation pour réitérer sa demande.

4.3 Heuristique d'insertion

Dans cette section, nous présentons l'heuristique que nous avons implémentée au coeur du module online et qui permet de répondre efficacement en temps réel aux requêtes des utilisateurs. Contrairement aux heuristiques de la littérature (section 3.2.4), elle offre une diversité de solutions à l'utilisateur et répond ainsi aux attentes d'un système de transport en commun à la demande. Elle permet également d'effectuer une optimisation sur les points de prise en charge et de dépose des utilisateurs.

L'algorithme 1 présente le pseudo-code de l'heuristique d'insertion. L'heuristique reçoit en entrée la requête utilisateur et le nombre maximum de noeuds sur lesquels une proposition peut être faîte à ce dernier. Toutes les positions d'insertion possibles sont d'abord recherchées (L1) puis testées pour ne garder que les insertions faisables (L2). Les insertions sont ensuite groupées

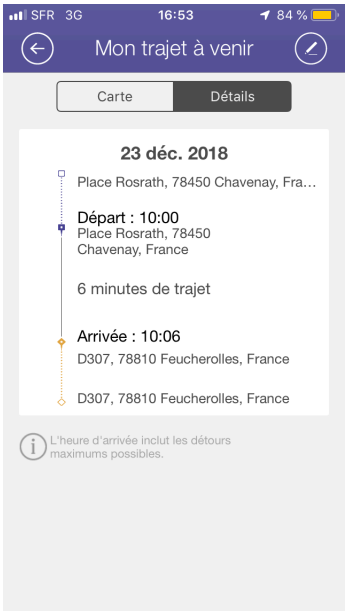


Figure 4.3 – Capture écran de l'écran de détails du trajet après paiement du client.

en fonction de l'heure de pickup ou dropoff associée (L3). Les insertions pouvant représenter des noeuds de pickup et dropoff différents, un filtre écartant potentiellement des insertions à des noeuds trop lointains de l'utilisateur est réalisé indépendamment sur chaque groupe (L4). Les insertions restantes de chaque groupe sont ensuite comparées entre elles et la meilleure de chaque groupe (au sens de l'objectif choisi) est conservée (L5). Ces multiples insertions s'échelonnant dans le temps sont ensuite proposées à l'utilisateur qui est libre de choisir celle qu'il désire. L'heuristique d'insertion permet de répondre aux besoins du système de transport en commun de Padam et de créer des tournées de qualité. Les sections suivantes analysent en détails chaque composante de l'heuristique.

Algorithm 1 Heuristique d'insertion

Entrée : Requête client R, nombre de noeuds à tester MNN, distance maximale aux noeuds MDN

Sortie : Liste des propositions client (peut être vide)

1: ListePositions ← $ChercheePositionPossibles$(R)
2: ListeInsertions ← $TesteInsertionsPossibles$(ListePositions, MNN, MDN)
3: DictGroupe← $GroupeInsertionsParIntervalle$(ListeInsertions) // Dictionnaire associatif ayant en clé un identifiant de groupe et en valeur toutes les insertions de ce groupe
4: DictGroupe ← $FiltreNoeuds$(DictGroupe)
5: ListeMeilleursInsertions ← $TrouveMeilleurInsertionsParIntervalle$(DictGroupe)
return ListeMeilleursInsertions

4.3.1 Recherche des positions d'insertion

La première étape de l'heuristique consiste à chercher les positions d'insertion faisables (ligne 1 de l'algorithme 1). Dans une tournée, une insertion est représentée par la position d'insertion du pickup et celle du dropoff, le dropoff étant évidemment positionné après le pickup. L'objectif est de trouver toutes les positions d'insertion possibles pour chaque tournée. La figure 4.4 fournit un exemple d'une tournée type. Le véhicule commence sa tournée au noeud

initial (position 0), attend quelque temps à son parking de départ puis va chercher à l'heure convenue deux utilisateurs au noeud 1. Ces derniers sont ensuite déposés successivement aux noeuds 2 et 3. Il n'y a aucune pause ou stationnement entre ces noeuds car les utilisateurs ne doivent pas attendre à bord d'un véhicule. On appelle "job" une suite de noeuds d'une tournée où le véhicule n'est pas vide (i.e il y a au moins un utilisateur à l'intérieur). Après avoir terminé son premier job, le véhicule stationne quelque temps avant d'aller se rendre au noeud 4 pour démarrer son deuxième job. Dans le cas présent, on parle de temps de latence car le temps entre la fin du job 1 et le début du job 2 est plus grand que le trajet pour se rendre du noeud 3 au noeud 4. La façon dont les positions d'insertion sont déterminées diffère entre les cas intra-job et inter-jobs.

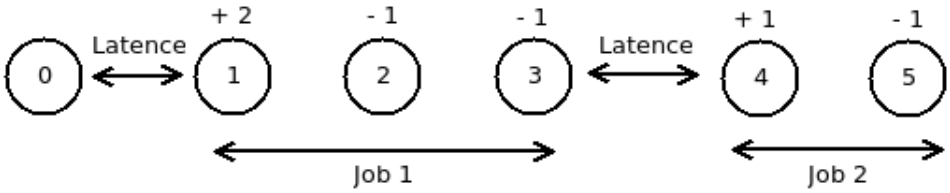


Figure 4.4 – Exemple d'une tournée avec deux jobs et de la latence entre les jobs

La figure 4.5 présente le comportement implémenté intra-job, c'est à dire lorsque la position de pickup se trouve à l'intérieur d'un job. Le job choisi est le job 1 de la figure 4.4. Chaque croix bleue représente une position possible pour le pickup. La première position se situe directement avant le noeud 1 de telle sorte qu'il n'y ait aucune latence entre le pickup et le noeud 1. Pour chaque position possible du pickup, un arc pointillé indique une possible position du dropoff. Toutes les positions possibles du dropoff sont considérées.

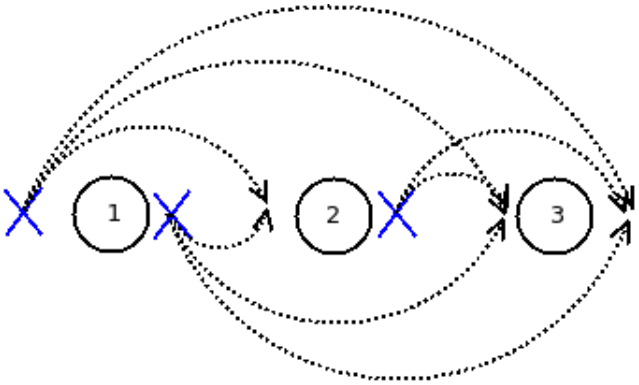


Figure 4.5 – Positions d'insertions testées sur un job avec trois noeuds

La figure 4.6 présente quant à elle le comportement implémenté lorsque les positions de pickup et de dropoff sont entre deux jobs (inter-jobs). L'exemple choisi est l'espace entre les jobs 1 et 2 de la figure 4.4. Chaque croix représente une heure d'insertion pour le pickup, l'heure de dropoff étant directement déduite car ce dernier est placé juste après le pickup. La première croix bleue représente la première heure d'insertion possible pour le pickup, calculée à partir du temps de trajet avec le noeud de la place 3. Les autres insertions sont ensuite testées toutes les RT minutes, RT étant un paramètre à fixer. Cette répartition temporelle des heures d'insertion permet de tester plusieurs insertions durant un intervalle de temps qui peut être grand, ce qui donnera au client final un plus grand choix. Le comportement est identique lors d'une insertion en fin de tournée, l'heure de la place 4 étant celle de l'heure maximale de fin de la tournée.

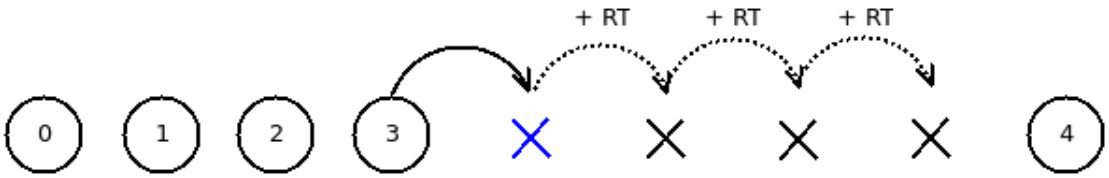


Figure 4.6 – Insertions testées lorsqu'il y a de la latence entre 2 places du véhicule.

Ces deux cas de figure (intra et inter-jobs) permettent de déterminer toutes les positions d'insertion d'une tournée. La procédure de recherche est réalisée pour chacune des tournées.

L'ensemble des positions d'insertion trouvé pour l'ensemble des tournées est ensuite restreint en fonction de l'heure de service (RH) désirée par l'utilisateur. En effet, l'heure de l'utilisateur constitue une indication sur son souhait réel et il est susceptible d'accepter des propositions qui peuvent être distantes de plusieurs dizaines de minutes. Il est cependant peu probable qu'il accepte des propositions distantes de plusieurs heures de l'heure demandée. Il est donc nécessaire de chercher des propositions dans un intervalle déterminé autour de l'heure requise mais il n'est pas pertinent de tester des insertions qui sont à plusieurs heures de l'heure demandée, particulièrement si le service sous-jacent s'étend sur toute la journée. En conséquence, si le client est PO (resp. DO), l'heure de pickup (resp. dropoff) associée à une insertion doit être dans l'intervalle $[\max(h - DB, t), h + DA]$ ou DB et DA sont des paramètres ajustables par l'opérateur de transport et t l'heure actuelle (important pour les requêtes en temps réel). Ces paramètres sont déterminés en fonction de l'expérience et des besoins du transporteur et permettent d'économiser un précieux temps de calcul car ils réduisent significativement le nombre des insertions qui seront testées dans l'étape suivante.

4.3.2 Test des insertions faisables

Une fois les positions d'insertion déterminées, l'étape suivante consiste à déterminer si elles sont faisables (L 2 de l'algorithme 1). Chaque insertion est associée à une position et à une paire de noeuds de pickup et dropoff. Les valeurs des paramètres MNN et MDN (L2) déterminent les paires de pickup et dropoff qui forment des insertions de la manière suivante : étant données les adresses de départ et d'arrivée des utilisateurs, les MNN noeuds les plus proches de l'adresse de départ et les MNN noeuds les plus proches de l'adresse d'arrivée sont déterminés. Parmi ces noeuds, seuls les noeuds à une distance de moins de MDN mètres sont conservés. Il ne reste donc que N_p noeuds de pickup et N_d noeuds de dropoff qui peuvent former $N_p * N_d$ paires de pickup/dropoff. Ce sont toutes ces paires qui seront testées pour chaque position d'insertion déterminée à l'étape précédente. Très souvent, MDN est de telle sorte que $N_p = N_d = MNN$ donc que le nombre total des paires pickup/dropoff testées est MNN^2. Il est donc important de garder une valeur raisonnable pour MNN afin d'éviter que le temps de réponse ne soit trop lent. Nous limiterons cette valeur à 10 dans les expérimentations.

Pour être faisable, une insertion doit respecter les contraintes du véhicule, des utilisateurs déjà insérés et du nouvel utilisateur : capacité, fenêtre horaire et temps de trajet maximum (voir 3.2.1 pour plus de détails).

4.3.3 Groupement des insertions

Une fois les différentes insertions faisables déterminées, elles sont regroupées en fonction des horaires associés (L3 de l'algorithme 1). L'établissement de plusieurs groupes sert de base

aux étapes suivantes de filtre et va permettre à l'utilisateur de disposer d'un choix multiple de propositions.

Tout d'abord, un ensemble d'intervalles disjoints et contigus sont créés entre l'heure de début et de fin du service. La figure 4.7 donne un exemple sur un service de 2h15. Les intervalles sont de même taille et le découpage est toujours le même quelle que soit l'heure demandée par le client. Ceci permet de garder une stabilité dans l'offre si un même client décide par exemple de réserver à 10 minutes d'intervalle. Une trop grande variation des propositions en fonction de l'heure de réservation du client serait mal comprise.

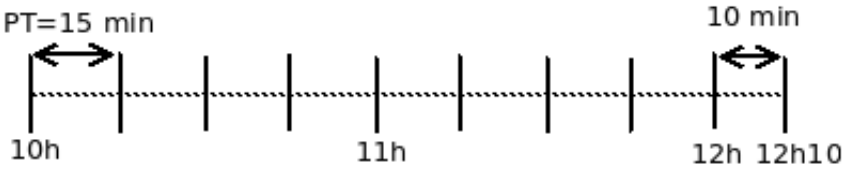


Figure 4.7 – Découpage du temps pour un service commençant à 10h et finissant à 12h10. *PT* est un paramètre libre qui fixe la taille des intervalles.

Une fois le découpage effectué, chaque insertion faisable va être associée à un intervalle par son heure d'intérêt, c'est-à-dire l'heure de pickup associée à l'insertion si le client est PO et l'heure de dropoff dans le cas contraire. La figure 4.8 illustre ce point avec le même découpage que la figure 4.7.

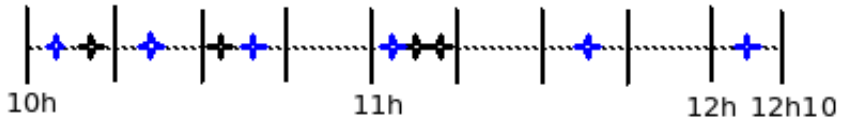


Figure 4.8 – Groupement et sélection des insertions par intervalle suivant l'heure d'intérêt. Chaque insertion est représentée par une étoile, la meilleure insertion du groupe étant colorée en bleu.

4.3.4 Filtrage des noeuds

L'étape précédente a permis de trier par intervalle de temps toutes les insertions faisables qui peuvent potentiellement être proposées à l'utilisateur. Ces insertions peuvent être associées à des paires de pickup/dropoff différentes de la paire "standard". La paire standard est définie comme la paire contenant le pickup et le dropoff standards. Le noeud de pickup (resp. dropoff) standard est défini comme le noeud le plus proche de l'adresse de départ (resp. arrivée) renseignée par l'utilisateur. Un filtre sur les noeuds (L4 de l'algorithme 1) est réalisé indépendamment sur chaque groupe d'insertions de la manière suivante : lorsque au moins une des insertions du groupe est associée à la paire standard, toutes les insertions qui ne représentent pas la paire standard sont éjectées. Cela signifie que si la paire standard est présente dans le groupe, toutes les autres paires pickup/dropoff présentes dans l'intervalle sont enlevées. Si elle n'est pas présente, aucune insertion n'est enlevée et le filtre n'a eu aucun effet sur le groupe. La présence de ce filtre sur les noeuds indépendamment de toute fonction objectif implique que des insertions avec des noeuds non standard peuvent être écartées au profit d'insertions moins bonnes avec noeuds standards. La raison d'être de cette limitation est avant tout d'assurer la transition avec le mono-noeud (système avec $MNN = 1$ qui équivaut à utiliser seulement les noeuds standards) qui fût le système par défaut jusqu'à très récemment. Pour ne pas perturber

l'usage régulier des clients, les noeuds les plus proches sont proposés dès que possible. Toutefois, cette politique n'est pas la seule que nous avons implémentée. Nous avons également implémenté une version où aucun filtre n'est réalisé à cette étape, ce qui équivaut à dire que la fonction à la ligne 4 de l'algorithme 1 est la fonction identité. Une comparaison des deux politiques de filtrage sera présentée dans les expérimentations.

4.3.5 Recherche des meilleures insertions

L'étape finale de l'heuristique consiste à déterminer pour chaque groupe la meilleure insertion qui sera ensuite proposée à l'utilisateur (L5 de l'algorithme 1). Comme pour l'étape précédente, le filtre s'applique indépendamment sur chaque groupe. Pour chaque insertion une valeur objectif est calculée, qui correspond à la valeur de la fonction objectif de la solution si l'insertion est réalisée. Voici les différentes métriques que nous avons utilisées en tant que fonction objectif :

TD A minimiser. Présenté section 3.2.1. L'idée derrière cette métrique est de créer des tournées qui servent l'ensemble actuel des requêtes en un temps minimal dans le but d'insérer plus naturellement les futures requêtes.

SK A maximiser. Pour chaque noeud visité par le véhicule, on définit l'écart au maximum comme étant la différence entre l'heure maximale possible du noeud et l'heure d'arrivée actuelle. SK est défini comme la somme de l'écart au maximum de toutes les places de toutes les tournées. L'idée est de laisser le plus de marge possible aux différents points d'arrêt des véhicules pour pouvoir faciliter l'insertion des futures requêtes. La contrainte imposée par l'heure d'arrivée maximale des noeuds est en effet la plus restrictive et souvent la cause de refus des insertions.

GA A minimiser. Cette métrique représente la somme des gammas effectifs de chaque requête utilisateur insérée dans les tournées. Le gamma effectif est le rapport entre le temps passé par le client dans le véhicule et son temps de trajet direct. L'idée est de créer des tournées avec le moins de détours pour les clients.

OB A minimiser. Un score est associé à chaque segment d'une tournée, un segment étant un trajet entre 2 positions (pickup/dropoff/parking) successives. Pour chaque segment, on divise le temps de trajet du segment par le nombre de passagers dans le véhicule durant ce trajet. L'objectif est la somme sur tous les segments de ce score sur toutes les tournées. L'idée est de favoriser la mutualisation des requêtes pour transporter le plus de personnes possibles par segment.

La meilleure insertion de chaque groupe est celle qui a le meilleur objectif, l'objectif étant une des métriques présentée ci-avant. Lorsque plusieurs insertions du même groupe sont équivalentes, elles sont toutes sélectionnées. Cette étape aboutit donc à une liste d'insertions réparties dans le temps qui correspondent aux insertions proposées à l'utilisateur (cf image 4.2).

4.4 Processus de validation

Après avoir choisi une proposition, une étape de validation est mise en oeuvre afin d'assurer que la proposition faite au client est toujours faisable. Pour ce faire, l'heuristique d'insertion exposée ci-avant est relancée, mais uniquement sur le véhicule concerné par la proposition choisie. Si une insertion faisable dont les heures de pickup et dropoff sont distantes de moins de 5 minutes par rapport aux heures de pickup et dropoff proposées est trouvée, le système

considère que la première proposition est toujours valide. Le client est inséré et la procédure est transparente à ses yeux. Dans le cas contraire, de nouvelles propositions lui sont faites comme sur l'image 4.2. Cette situation étant rare sur les services réels associés aux instances de 3.3.2, aucune de nos simulations ne prendra en compte ce comportement : tous les clients réserveront les uns à la suite des autres, sans risque de "collision". Ceci est assuré par le fait que le simulateur est construit sur une architecture basée évènement où les évènements sont exécutés à la suite (cf chapitre 7).

Supposons désormais que le client valide une proposition. L'image 4.3 présente une capture d'écran des détails relatifs à son trajet après validation avec les informations relatives au lieu de prise et aux horaires de prise en charge. Supposons que le client ait choisi une proposition ayant h_p et h_d comme heures de pickup et dropoff respectives. Une fenêtre horaire sur le pickup (TW_p) et sur le dropoff (TW_d) sont calculées de la manière suivante :

$$TW_p = [h_p - PWB, h_p + PWA]$$
$$TW_d = [h_d - DWB, h_d + DWA]$$

où les paramètres PWA, PWB, DWA et DWB sont fixés après discussion entre Padam et l'opérateur de transport. Les valeurs de ces paramètres pour nos instances sont données en section 3.3.2. Ces fenêtres horaires deviennent des contraintes dures associées à l'utilisateur. Combinées à la contrainte de temps de trajet maximal (cf section 3.2.1), elles permettent d'assurer que des insertions ultérieures ne nuiront pas à la qualité de service de l'utilisateur.

4.5 Expérimentations

L'objectif de cette section est d'étudier le comportement de l'heuristique d'insertion présentée ci-avant. Les expérimentations utiliseront les instances présentées section 3.3.2. Les valeurs de DA et DB (4.3) sont de 60 minutes chacune. Sauf indication contraire, $MNN = 1$ et $MDN = 600$ mètres. Toutes les simulations sont menées sur une machine contenant 32GB de RAM et un processeur quadri-core Intel Xeon E3-1245 cadencé à 3.40GHZ. Le module est implémenté entièrement en Python/Cython [1]. Les métriques d'intérêt que nous étudierons sont les suivantes :

PSR : pourcentage de requêtes servies

RTD : durée totale des tournées divisée par le nombre de requêtes servies

MAXWKT : temps de marche (minutes) maximal effectué par les clients. Le temps de marche d'un client est la somme des temps de marche aux noeuds de pickup et dropoff du client.

MEANWKT : temps de marche moyen de tous les clients. Le temps de marche d'un client est la somme des temps de marche aux noeuds de pickup et dropoff du client.

4.5.1 Impact PT et RT

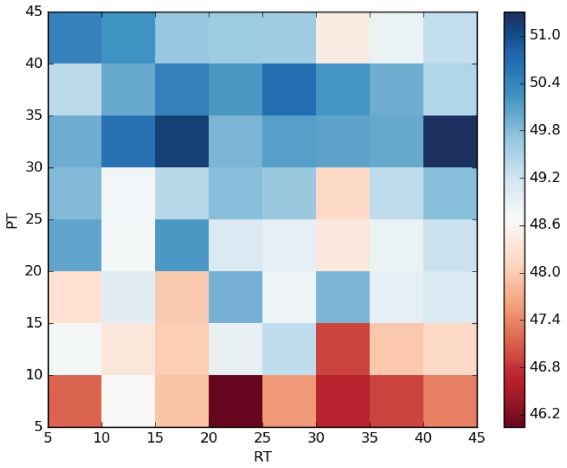
Nous étudions ici l'influence des deux paramètres PT et RT sur PSR. Rappelons que PT défini section 4.3.3 détermine la taille des intervalles de proposition et RT défini section 4.3.1 détermine le nombre d'insertions testées lorsqu'il y a de la latence entre deux noeuds. Pour

1. https ://cython.org/

ce faire, nous avons fait varier les valeurs des paramètres entre 5 et 45 minutes, par pas de 5 minutes. Une simulation a été réalisée sur chaque instance avec chaque configuration de paramètres (valeurs pour PT et pour RT). Nous présenterons les résultats séparément pour chaque groupe.

Groupe A

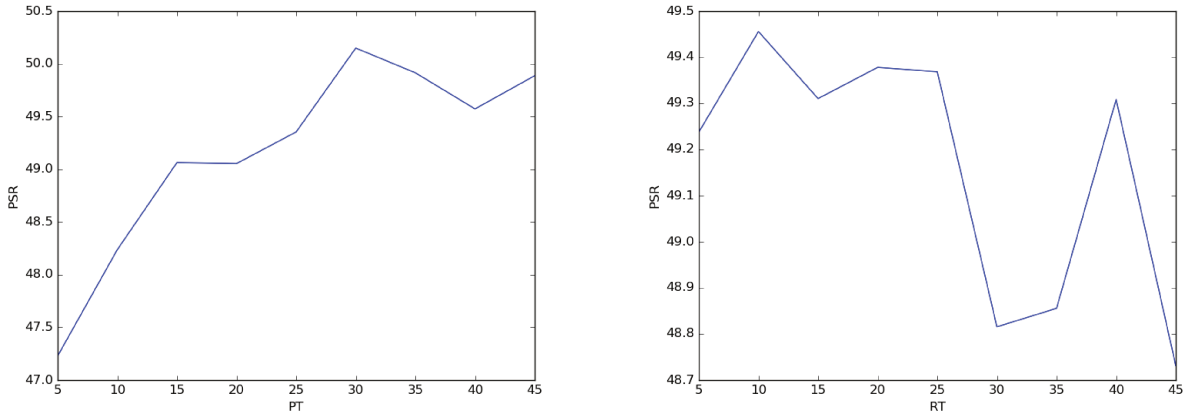
La figure 4.9a présente la carte thermique ("heat map") associée aux simulations pour le groupe A. Une couleur proche du bleu indique une valeur élevée tandis qu'une couleur proche du rouge indique une valeur faible.



(a) Carte thermique du groupe A

Le pourcentage de requêtes servies varie entre 46 et 51%, ce qui montre qu'au moins un des deux paramètres a une influence non négligeable sur les performances. On observe que les performances s'améliorent au fur et à mesure que PT augmente (couleur plus orientée vers le bleu), en particulier lorsque $PT > 25$ minutes. Les meilleurs performances sont obtenues lorsque $PT = 30$ minutes. Le graphique 4.10a confirme ces observations : les performances augmentent jusqu'à atteindre 30 minutes puis redescendent légèrement. Les valeurs de $PT > 30$ minutes ont de moins bonnes performances car la fréquence de proposition à l'utilisateur (qui accepte une proposition dans les 20 minutes autour de son heure demandée) est trop faible, conduisant à de plus nombreux refus. Une plus petite valeur quant à elle augmente la cadence de proposition pour un utilisateur mais augmente également la probabilité qu'une insertion de moins bonne qualité soit proposée, ce qui peut dégrader les tournées et nuire à l'insertion de futurs utilisateurs. Concernant RT, la figure 4.9a semble indiquer que ce dernier a un impact beaucoup plus faible sur les performances en comparaison avec PT. Ce constat est confirmé par le graphe 4.10b. On observe en effet que le pourcentage de requêtes servies varie entre 48.7 et 49.5%, ce qui constitue une très faible variation.

Il est donc important d'utiliser des valeurs élevées pour PT, la valeur de RT étant moins importante sur les performances globales du système. Le meilleur couple de valeurs pour le groupe A est $PT = 30$ minutes et $RT = 40$ minutes.



(a) PT (b) RT

Figure 4.10 – Impact de PT et de RT sur le groupe A. PSR (en ordonnée) représente le pourcentage moyen de requêtes servies sur toutes les instances pour toutes les valeurs de PT (resp. RT) lorsque RT (resp. PT) est fixé

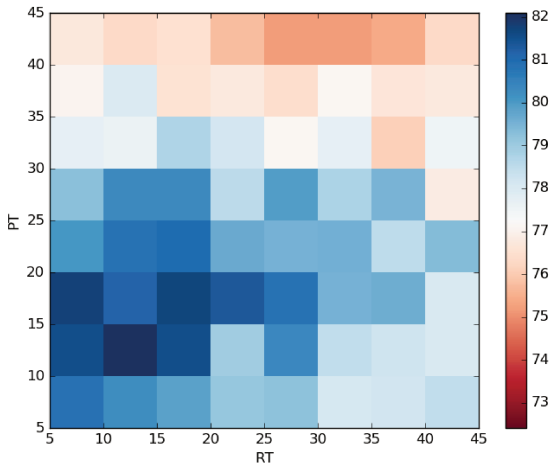
Groupe B

La figure 4.11a présente la carte thermique ("heat map") associée aux simulations pour le groupe B. Sur le groupe B, on observe clairement que les meilleurs résultats sont obtenus lorsque les valeurs de RT et PT diminuent, avec un maximum aux environs de 15 minutes. Lorsque PT est plus grand que 25 minutes, le pourcentage de requêtes servies diminue systématiquement et de manière significative. La figure 4.12a permet de mettre en valeur ces observations : le maximum est atteint pour $PT = 15$ minutes et les performances diminuent au fur et à mesure que PT augmente. Ce comportement est similaire au cas du groupe A, avec une valeur maximale différente. On peut également remarquer que l'impact de PT sur le pourcentage de requêtes servies est significatif, avec des valeurs variant de 80 à 74%. Concernant RT, on observe une diminution lorsque $RT > 25$ minutes mais cette diminution est moins marquée lorsque $PT <= 25$ minutes. Ceci montre que l'augmentation de RT entraîne une diminution de la performance et que l'impact de RT est plus faible que PT. Ces observations sont plus explicites sur la courbe 4.12b, où les valeurs de PSR varient entre 79 et 76.5% et diminuent lorsque RT augmente.

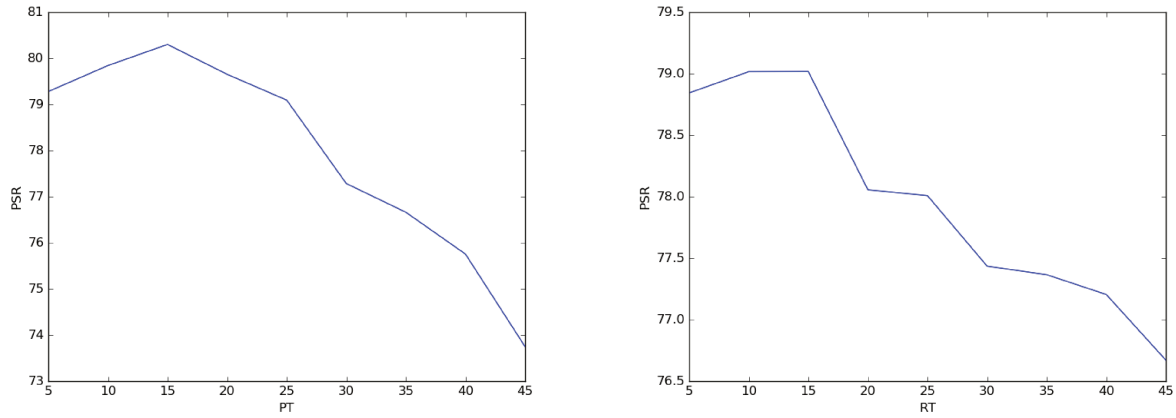
Il est donc important de ne pas utiliser des valeurs trop élevées pour ces paramètres, particulièrement pour PT qui a un plus grand impact négatif sur les performances. Le meilleur couple de valeurs pour le groupe B est $PT = 10$ minutes et $RT = 10$ minutes.

Comparaison entre les deux groupes

Les deux groupes présentent un comportement très différent. Sur le groupe A, il est préférable d'utiliser des valeurs élevées pour PT et RT tandis que les valeurs doivent être plus faibles sur le groupe B. PT a un impact significatif et similaire sur les deux groupes : des valeurs trop petites ou trop grandes nuisent aux performances. RT en revanche a un impact plus faible sur le groupe B et quasi-inexistant sur le groupe A pour des valeurs inférieures à 45 minutes.



(a) Carte thermique du groupe B



(a) PT

(b) RT

Figure 4.12 – Impact de PT et de RT sur le groupe B. PSR (en ordonnée) représente le pourcentage moyen de requêtes servies sur toutes les instances pour toutes les valeurs de PT (resp. RT) lorsque RT (resp. PT) est fixé

4.5.2 Choix de la fonction objectif

Dans cette section, nous allons comparer les métriques présentées en section 4.3.5. Les critères considérés pour comparer une métrique à une autre seront PSR et RTD.

Groupe A

Le tableau 4.1 montre les résultats des différentes simulations sur les instances du groupe A. Une valeur faible pour RTD indique que de nombreuses requêtes ont été servies avec des coûts de fonctionnement faibles ou raisonnables. On observe que le critère TD (durée des tournées) obtient les meilleurs performances sur 9 instances. Le critère OB se place dans la plupart des cas en deuxième position, ce qui est confirmé par les performances moyennes. TD est donc de loin le meilleur objectif, suivi par OB. Les deux autres objectifs proposés donnent des performances significativement moins bonnes. De plus on observe que le meilleur objectif en terme de PSR est aussi toujours le meilleur en terme de RTD, même sur l'instance A_177 où SK est l'objectif

obtenant les meilleures performances.

	PSR				RTD			
VE	TD	SK	GA	OB	TD	SK	GA	OB
A_120	**77.5**	70.0	74.17	75	**1034**	1272	1298	1156
A_145	**59.31**	56.55	51.03	57.93	**1224**	1369	1501	1298
A_159	**66.67**	55.98	54.09	62.27	**933**	1255	1287	1028
A_177	45.76	**47.46**	41.24	45.76	1302	**1300**	1539	1328
A_216	**46.76**	41.6	37.5	42.13	**1096**	1269	1427	1204
A_233	**60.94**	44.21	51.50	58.37	**755**	1116	976	802
A_271	**37.64**	32.47	35.42	35.42	**1086**	1303	1185	1146
A_288	**36.81**	32.99	29.86	33.33	**1048**	1202	1344	1143
A_288b	**52.78**	44.44	47.92	47.22	**731**	892	824	824
A_360	**28.89**	26.11	28.33	26.67	**1065**	1222	1113	1146
Avg.	**51.31**	45.19	45.11	48.41	**1028**	1220	1250	1128

Tableau 4.1 – Comparaison des différentes métriques pour le groupe A (définies section 4.3.5)

Groupe B

Les résultats pour le groupe B sont présentés dans le tableau 4.2. TD est le meilleur dans 6 instances sur 9. OB obtient le maximum de PSR sur 3 instances. A l'exception de B_200 et B_1011, la métrique qui permet de servir le plus de requêtes est aussi celle qui a un RTD le plus faible. Les métriques SK et GA d'une part et TD et OB d'autre part ont des performances très similaires entre elles, le couple TD/OB étant beaucoup plus efficace.

	PSR				RTD			
VE	TD	SK	GA	OB	TD	SK	GA	OB
B_200	89.0	90.0	**92.5**	88.0	448	579	568	**446**
B_301	90.03	90.69	89.04	**91.03**	466	567	556	**454**
B_398	**89.19**	86.93	88.44	87.69	**412**	488	497	413
B_498	**87.15**	81.93	80.72	86.94	**390**	496	489	398
B_607	**82.04**	76.44	77.10	78.91	**368**	462	440	376
B_709	**78.70**	70.66	69.68	78.56	**348**	440	442	350
B_802	**76.93**	66.33	67.58	72.57	**330**	422	411	354
B_909	**74.56**	64.54	64.87	**74.56**	320	383	388	**317**
B_1011	71.22	61.52	61.13	**72.21**	**289**	367	369	295
Avg.	**82.09**	76.56	76.78	81.16	**375**	467	462	378

Tableau 4.2 – Comparaison des différentes métriques pour le groupe B

Synthèse des résultats sur les deux groupes

On observe que TD est le meilleur objectif pour les deux groupes, OB étant en deuxième position. Ce dernier a des performances très similaires à TD sur le groupe B en étant meilleur ou égal sur un tiers des instances. Le critère TD est un objectif simple et efficace et ces

expérimentations montrent qu'il est difficile de trouver un meilleur objectif mono-attribut. Les résultats montrent également que sur presque toutes les instances, la métrique qui obtient les meilleurs résultats en terme de PSR est aussi celle qui obtient le plus petit RTD. Ceci confirme qu'il existe une corrélation forte entre le fait de servir de nombreuses requêtes et d'avoir des tournées optimisées en terme de déplacement des véhicules. Lors de chaque nouvelle requête, le nombre de requêtes que traite l'algorithme online est fixé. TD minimise ainsi directement cette quantité RTD avec les informations partielles dont il dispose.

4.5.3 Intégration des noeuds dans l'optimisation

L'objectif de cette section est d'étudier les performances de l'heuristique lorsque plusieurs noeuds peuvent être proposés au client, c'est à dire $MNN > 1$. Nous diviserons l'analyse en deux parties : une avec le filtre multi-noeuds activé et l'autre sans ce filtre. Pour les expérimentations, nous rappelons que la distance maximum autorisée MDN est fixée à 600 mètres pour les deux groupes d'instances. Cette valeur est fixée en accord avec les transporteurs respectifs. Les expérimentations de cette section présenteront l'évolution des différentes métriques (PSR, RTD, $MAXWKT$ et $MEANWKT$) en fonction de MNN (nombre de noeuds autorisés durant une recherche client). Pour chaque groupe, les valeurs des métriques sont obtenues en faisant la moyenne des valeurs de chaque instance du groupe.

Avec filtre sur les noeuds

La figure 4.17 présente pour le groupe A plusieurs métriques d'intérêt en fonction du nombre de noeuds autorisés pour la recherche (MNN). On observe en figure 4.13 que le nombre de requêtes servies est croissant en fonction de MNN et passe de 51% à près de 60% lorsque $MNN = 10$. C'est une augmentation très importante qui correspond à une amélioration relative de plus de 15%. La figure 4.14 montre que le coût moyen de transport d'un client diminue lorsque MNN augmente, jusqu'à atteindre une diminution de 10%. Cela indique que les tournées sont mieux optimisées lorsque MNN augmente car plus de requêtes sont servies avec un coût moindre. L'augmentation de MNN apporte donc un gain très important sur les performances du système.

Il est néanmoins nécessaire de mettre en perspective ces résultats avec la qualité de service offerte au client que l'on peut évaluer via les temps de marche qui lui sont imposés. La figure 4.15 nous montre que le temps de marche total maximum augmente de manière significative lors du passage de 1 à 2 qui correspond à l'activation du multi-noeuds. Précisons que le temps de marche n'est pas nul dans le cas où $MNN = 1$ car certaines requêtes des instances du groupe A ont une adresse de pickup et/ou dropoff qui n'est pas nécessairement l'adresse d'un noeud du maillage. Le temps de marche maximal augmente ensuite très peu (environ de 14 à 17 minutes) entre 2 et 10 noeuds. Cette augmentation soudaine est due à un faible nombre de client demandant dans des zones peu denses où le deuxième noeud le plus proche est loin du premier. Ceci semble être confirmé figure 4.16 où le temps de marche moyen augmente progressivement et ne dépasse pas 3 minutes. Ceci nous apprend que le temps total de marche est acceptable pour la majorité des clients et que les temps de marche maximaux présentés figure 4.15 concernent une minorité de requêtes.

La figure 4.22 présente les résultats dans le cas du groupe B.

Comme dans le cas du groupe A, on observe (figure 4.18) que le nombre de requêtes servies augmente significativement, approchant en moyenne des 94%. On observe de même figure 4.19

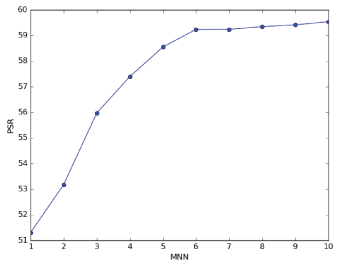


Figure 4.13 – PSR

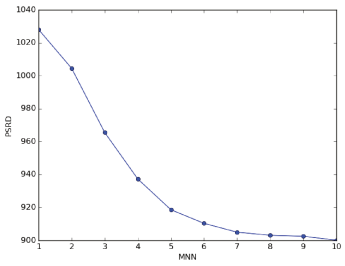


Figure 4.14 – RTD

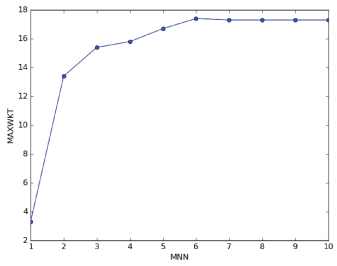


Figure 4.15 – MAXWKT

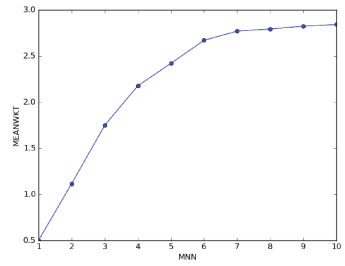


Figure 4.16 – MEANWKT

Figure 4.17 – Evolution des métriques d'intérêt en fonction du nombre de noeuds autorisés (MNN) pour le groupe A. Filtre multi-noeuds activé

que la procédure d'optimisation est plus efficace lorsque MNN augmente. Un bond en terme de temps de marche maximal est aussi observé figure 4.20 lors du passage de $MNN = 1$ à $MNN = 2$. Contrairement au groupe A, le temps maximal continue d'augmenter jusqu'à presque 20 minutes, ce qui est du à la plus grande densité du maillage. Le temps moyen de marche en revanche reste très faible (moins de 2 minutes 30), comme l'indique la figure 4.21. Comme pour le groupe A, le multi-noeuds apporte un gain significatif aux performances du système. Cependant, il est probablement nécessaire de diminuer la distance maximale autorisée sur le groupe B car les trajets demandés par les clients sont plus courts que dans le groupe A. En effet, un temps de marche trop long en comparaison avec le trajet que désire effectuer le client est fortement préjudiciable à la qualité de service.

Sans filtre sur les noeuds

Nous allons comparer ici les résultats obtenus en enlevant le filtre multi-noeuds (dont l'action principale est de retirer les noeuds non standard) par rapport aux résultats précédents. La figure 4.27 expose les résultats sur le groupe A.

Les figures 4.23 et 4.24 montrent le gain très important apporté en enlevant le filtre. Ainsi, $MNN = 3$ sans filtre permet d'obtenir des meilleures performances que $MNN = 10$ avec filtre. On observe en figure 4.25 que l'augmentation du temps total de marche maximal n'est pas très importante, ce qui n'est pas très surprenant car les noeuds considérés sont les mêmes dans les deux cas et seule une requête acceptée sur un noeud lointain suffit à faire augmenter la statistique. En revanche, la figure 4.26 montre que le temps moyen est beaucoup plus important lorsque le filtre est enlevé. C'est également attendu car la meilleure proposition dans

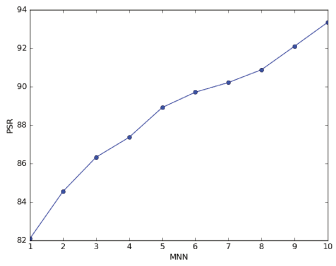


Figure 4.18 – PSR

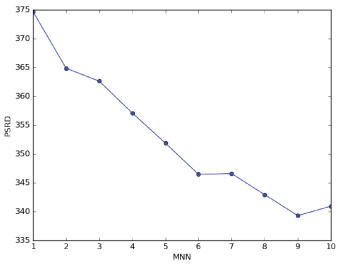


Figure 4.19 – RTD

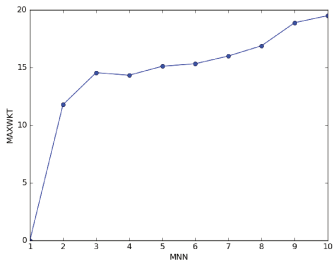


Figure 4.20 – MAXWKT

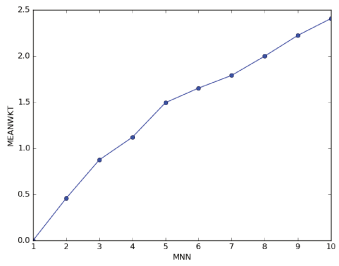


Figure 4.21 – MEANWKT

Figure 4.22 – Evolution des métriques d'intérêts en fonction du nombre de noeuds autorisés (MNN) pour le groupe B. Sans filtre multi-noeuds

un intervalle de temps a plus de chances de concerner des noeuds plus éloignés lorsque le filtre est désactivé. Malgré le gain considérable apporté par le retrait du filtre, le transporteur ne peut choisir une valeur de MNN élevée sans évaluer l'impact utilisateur. Il est nécessaire de trouver un équilibre entre une meilleure performance du système et un temps de marche client raisonnable. La valeur de $MNN = 3$ est un candidat potentiel, en fournissant une augmentation significative des performances tout en limitant le temps de marche moyen à moins de 10 minutes (pickup + dropoff).

La figure 4.32 expose les résultats sur le groupe B. Comme pour le groupe A, on observe une amélioration significative des performances (cf figures 4.28 et 4.29). On observe notamment que le pourcentage de requêtes servies approche des 100% lorsque $MNN = 10$ et que le coût moyen de transport d'une personne est presque diminué par 2 lorsque MNN augmente. Comme dans le cas du groupe A, l'augmentation du temps de marche maximal est faible (figure 4.30). Cependant, la figure 4.31 montre que l'augmentation du temps moyen de marche est très marquée, étant multipliée environ par 4 pour chaque valeur de MNN. Etant donné la petite taille du territoire, le transporteur doit être prudent quant aux valeurs choisies pour MNN pour éviter des temps de marche trop grands (figure 4.29). L'apport du retrait du filtre sur les performances est beaucoup plus marqué sur RTD que sur PSR et les augmentations importantes du temps de marche moyen peuvent ne pas justifier le retrait du filtre sur le groupe B.

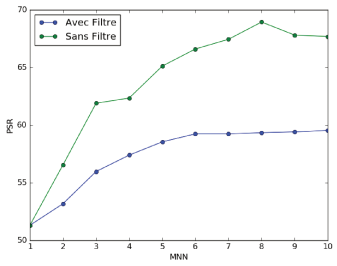


Figure 4.23 – PSR

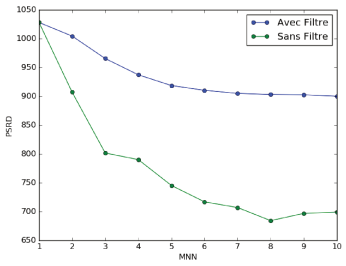


Figure 4.24 – RTD

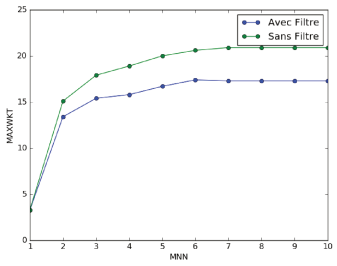


Figure 4.25 – MAXWKT

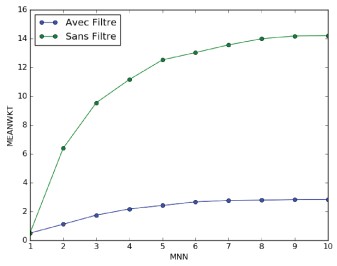


Figure 4.26 – MEANWKT

Figure 4.27 – Comparaison des métriques d'intérêts avec et sans filtre multi-noeuds pour le groupe A

Synthèse sur l'integration des noeuds dans l'optimisation

Nous avons vu que le multi-noeuds apporte un gain significatif aux performances de l'online tout en gardant les temps de marche moyens raisonnables. Ces résultats justifient largement l'utilisation du multi-noeuds sur des services réels. Le retrait du filtre permet d'augmenter encore de manière importante les performances, au prix toutefois d'une augmentation conséquente des temps de marche moyens. Ceci est particulièrement valable sur le groupe B où le maillage est plus dense que sur le groupe A. Avant d'enlever le filtre, il est nécessaire de prendre en compte les différentes métriques ayant trait aux performances du système et à la qualité de service pour déterminer une valeur de MNN qui offre un équilibre pertinent.

4.6 Conclusion

Dans ce chapitre, nous avons présenté la première contribution développée durant la thèse, à savoir le module online. Nous avons développé une heuristique multi-horaire capable de créer des tournées de qualité et de répondre aux besoins du système de transport en commun proposé par Padam. Nous avons étudié l'impact de ses différents paramètres et mis en exergue un comportement différent sur les deux groupes d'instances. En utilisant différentes fonctions objectifs, nous avons montré que la métrique de durée totale des tournées est le meilleur des objectifs testés et qu'il est difficile de trouver une métrique plus performante en tant qu'objectif. Enfin, nous avons intégré avec succès la prise en charge des noeuds du maillage dans l'heuristique et avons obtenu une amélioration significative des performances.

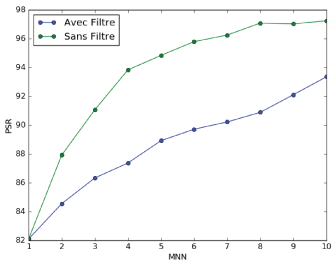


Figure 4.28 – PSR

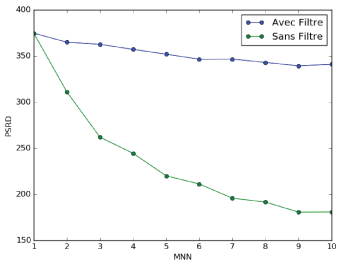


Figure 4.29 – RTD

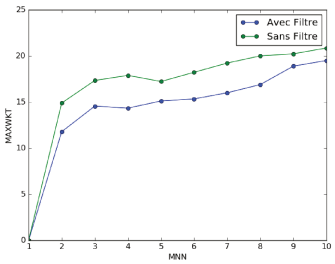


Figure 4.30 – MAXWKT

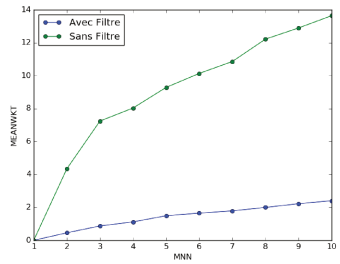


Figure 4.31 – MEANWKT

Figure 4.32 – Comparaison des métriques d'intérêts avec et sans filtre multi-noeuds pour le groupe B

Malgré la pertinence de l'heuristique, certaines requêtes ne recevant pas de réponse adéquate sont toujours rejetées. Pour pallier au mieux ce problème, le chapitre suivant présente deux heuristiques dites de "réinsertion" permettant d'insérer en temps réel des requêtes refusées par l'online.

Chapitre 5

Module de Réinsertion

5.1 Introduction

Dans le chapitre 4, nous avons décrit l'algorithme d'insertion que nous avons implémenté dans le module Online de Padam. La flotte dont nous disposons étant de taille finie, nous avons vu que certains clients peuvent ne pas recevoir de réponse adéquate. Dans certains cas, cela est du à l'aspect naïf de l'algorithme online qui se contente de tester toutes les insertions possibles en l'état actuel de la solution. Lorsqu'il ne trouve pas de solution satisfaisante, il est possible qu'une simple réorganisation de requêtes déjà insérées libère de la place pour insérer la nouvelle requête. Cette stratégie qui consiste à exploiter le voisinage d'une solution pour insérer une requête rejetée par l'algorithme Online est connue sous le nom de "réinsertion". Ce chapitre exploite ce levier d'optimisation pour améliorer les performances du système de transport à la demande chez Padam. La section 5.2 présente la problématique de réinsertion dans le contexte de Padam. Les sections 5.3 et 5.4 présentent deux méthodes de réinsertion que nous avons développées pour notre problème. Ces deux heuristiques sont basées sur des voisinages très différents, l'une étant basée sur la notion de Destruction/Réparation et l'autre sur le concept de Chaîne d'Ejections. Pour mesurer la performance des différentes heuristiques, nous comparerons le nombre de requêtes servies et la durée totale des tournées finales obtenues par rapport à l'algorithme online simple. Les expérimentations sont présentées en section 5.7 et réalisées sur les instances présentées en section 3.3.2. Une conclusion sera enfin présentée en section 5.10.

5.2 Présentation de la problématique

Dans le chapitre 4, nous avons explicité le fonctionnement de l'heuristique d'insertion principale du module online. Comme l'ont montré les expérimentations, certaines requêtes sont parfois refusées. La principale raison de ces refus est que l'heuristique ne peut trouver d'insertion faisable suffisamment proche de l'heure désirée par l'utilisateur. L'objectif du module de réinsertion est de trouver une insertion de bonne qualité dans un intervalle de temps autour de l'heure demandée par l'utilisateur lorsque le module online n'a pas trouvé de réponse satisfaisante. Contrairement au module online, le module de réinsertion peut déplacer les requêtes déjà insérées pour chercher des solutions dans un voisinage de la solution actuelle.

5.2.1 Données d'entrée et interaction avec le module online

Le module online a la possibilité d'appeler le module de réinsertion si il désire intensifier la recherche dans un créneau horaire important pour le client où aucune insertion n'a été trouvée. Le module de réinsertion reçoit les informations suivantes :

— La requête de l'utilisateur
— Un intervalle de temps
— Temps d'exécution maximum

La requête de l'utilisateur est détaillée en section 4.2.1 et contient les informations géographiques et temporelles de l'utilisateur. L'intervalle de temps fournit indique au module dans quel créneau horaire il doit trouver une insertion pour le client. Si le client est orienté pickup (resp. orienté dropoff), le module de réinsertion devra trouver une insertion avec une heure de pickup (resp. dropoff) contenue dans l'intervalle donné. Ce dernier est déterminé par le module online comme étant égal à $[RH - RW, RH + RW]$ où RW est un paramètre (en minutes) à déterminer et RH l'heure demandée par l'utilisateur. Cet intervalle est centré autour de RH pour se rapprocher au plus de l'heure demandée par l'utilisateur. Pour nos expérimentations, RW est égal à la déviation maximale autorisée par le client notée W (cf section 3.3.2). L'intervalle attendu par le module de réinsertion étant toutefois générique, le module online peut utiliser le module de réinsertion pour intensifier la recherche dans n'importe quel créneau horaire désiré.

Une fois le module de réinsertion terminé (au plus tard lorsque le temps d'exécution maximum est atteint), la meilleure insertion trouvée est envoyée au module online. Ce dernier dispose donc de l'insertion trouvée par le module de réinsertion dans le créneau horaire transmis et des insertions trouvées par sa propre heuristique d'insertion (section 4.3) en dehors de ce créneau horaire. Toutes les propositions sont ensuite envoyées au client comme dans le processus habituel du module online (section 4.2.2).

5.2.2 Contraintes et défis

Le problème auquel le module de réinsertion fait face peut se résumer de la manière suivante : étant donné une requête et un intervalle de temps, trouver une configuration des tournées qui permet d'insérer le nouvel utilisateur dans l'intervalle de temps de la meilleure des manières possibles. Pour trouver une solution dans l'intervalle demandé, le module de réinsertion dispose d'un nombre fini de tournées dans lesquelles il a la possibilité de déplacer les utilisateurs déjà insérés. Il doit cependant respecter leurs contraintes horaires (fenêtre horaire et gamma) et les contraintes de capacité des véhicules. De plus, il doit fournir une réponse rapide (quelques secondes) pour répondre aux contraintes temps réel qui lui sont imposées. Il est donc nécessaire de concevoir une heuristique pouvant explorer rapidement le voisinage d'une solution pour trouver des solutions efficaces.

5.2.3 Objectifs opérationnels et scientifiques

Le développement d'un module de réinsertion efficace est très important pour permettre de servir le plus de requêtes possible dans le cadre des services opérés par Padam. A notre connaissance, seul [Luo and Schonfeld, 2011] a travaillé sur le développement d'algorithmes de réinsertion pour le DARP dynamique (cf section 3.2.4). Les auteurs considèrent cependant une flotte infinie de véhicules et proposent une heuristique simple qui n'explore pas en profondeur le voisinage de la solution actuelle. De nombreuses pistes sont envisageables pour proposer des

heuristiques qui permettent de trouver des solutions de haute qualité en explorant intensivement le voisinage pendant le temps imparti. Le problème posé par la réinsertion présente donc un intérêt à la fois sur les plans opérationnel et scientifique.

5.3 Heuristique de Destruction/Réparation (HDR)

L'heuristique présentée dans cette section est inspirée des heuristiques de type LNS (Large Neighborhood Search). [Shaw, 1998] est le premier à avoir introduit l'idée sous forme d'une méta-heuristique. L'idée est de détruire un pourcentage important de la solution pour la reconstruire ensuite. Le voisinage créé est ainsi de très grande taille et seule une petite partie est visitée durant la recherche. L'objectif d'un grand voisinage est de pouvoir explorer une grande variété de zones de l'espace des solutions tout en évitant d'être bloqué dans des extrema locaux. Durant le reste de ce chapitre, nous appellerons HDR l'heuristique de réinsertion basée sur la notion de destruction/réparation. L'algorithme 2 présente le pseudo-code associé à l'algorithme HDR.

Algorithm 2 Heuristique de Réinsertion (HDR)

Entrée : solution actuelle s, nouvelle requête R, liste des opérateurs de destruction LDO, liste des opérateurs de construction LRO, temps maximum MT

Sortie : meilleure solution trouvée si elle existe

1: L← $EmptyList()$
2: **while** Temps écoulé $< MT$ **do**
3: op← $SelectDestructionOperateurs$(LDO)
4: n← $ChoisiNombreRequetes()$
5: req← $EnleveRequetes$(op, n)
6: req← $req \cup \{R\}$
7: $s_{new} \leftarrow ReinsertRequetes$(req, LRO,s)
8: **if** $feasible(s_{new})$ et $accept(s_{new})$ **then** $L \leftarrow L \cup s_{new}$;
9: **end while**
10: **if** L non vide **then** retourne $MeilleurSolution(L)$

L'algorithme 2 reçoit en paramètres la requête du nouvel utilisateur (R), les différents opérateurs de destruction et de réparation et le temps maximum. La première étape consiste à choisir les opérateurs de destruction (L3) qui permettront d'enlever certaines requêtes des tournées de la solution actuelle. Un nombre aléatoire de requêtes est ensuite choisi (L4) puis enlevé de la solution actuelle (L5). Ces 3 étapes correspondent à la phase de destruction de l'heuristique. Les requêtes ainsi enlevées sont ensuite regroupées avec la nouvelle requête R (L6) pour être ensuite réinsérées dans les tournées (L7) : c'est la phase de réparation. **Toutes** les requêtes doivent être réinsérées pour obtenir une solution faisable. Ces phases de destruction et de réparation sont répétées jusqu'à ce que le temps maximal MT soit écoulé. Les sections suivantes présentent en détails les étapes de destruction et de réparation.

5.3.1 Etape de Destruction

La première étape consiste à choisir un ensemble de requêtes à enlever de la solution actuelle. Pour ce faire, nous utiliserons 3 opérateurs de destruction différents. Chaque opérateur

sélectionne les requêtes à enlever parmi une liste LI de requêtes déjà insérées dans la solution. LI ne sera toutefois pas forcément égal à la totalité des requêtes de la solution. En effet, il n'est pas pertinent d'enlever une requête éloignée en temps de la nouvelle requête car il est peu probable que son retrait d'une tournée aura une influence sur l'insertion de la nouvelle requête R. Nous définissions ainsi une fenêtre horaire $[RH - W - T, RH + W + T]$ avec RH l'heure demandée par le client, W la déviation maximale acceptée par le client et T un paramètre libre. La liste LI est restreinte aux requêtes dont l'heure de pickup ou de dropoff sont à l'intérieur de la fenêtre horaire. La valeur de T est donc importante : si T est trop petit, LI sera trop restreinte et certaines requêtes pertinentes risquent de ne jamais être choisies. Si T est en revanche trop grand, presque toutes les requêtes peuvent être enlevées et beaucoup de temps peut ainsi être perdu, le phénomène s'amplifiant lorsque la taille de l'instance augmente.

En plus de cette liste de candidats, chaque opérateur reçoit aussi le nombre de requêtes k à choisir. k est généré aléatoirement à chaque itération entre k_{min} and k_{max}, qui sont 2 paramètres libres. A chaque itération, l'opérateur de destruction utilisé est choisi aléatoirement. Voici désormais la description des 3 opérateurs de destruction utilisés, inspirés des opérateurs de la littérature ([Shaw, 1998], [Ropke and Pisinger, 2006]) :

Random : Choisit aléatoirement k requêtes.

Worst : Choisit les k requêtes ayant les meilleurs "savings", le "saving" d'une requête étant la différence entre la valeur objectif de la solution actuelle et celle de la solution une fois la requête enlevée. Pour augmenter la diversification, l'opérateur est randomisé : toutes les requêtes de LI sont triées en ordre décroissant de "saving" dans une liste L. Un nombre aléatoire y entre 0 et 1 est tiré et la requête en position $\lfloor y^{p_r}|L| \rfloor$ (où $|L|$ est la taille de la liste L et p_r un paramètre) est choisie. Cette procédure est répétée jusqu'à ce que k requêtes aient été choisies.

Relatedness : Choisi aléatoirement une requête et sélectionne $k - 1$ requêtes similaires. La mesure de similarité entre les requêtes i et j est définie comme suit :

$$\frac{1}{2}\left(t_{p_i,p_j} + t_{d_i,d_j}\right) + \frac{1}{2}\left(|u_{p_i} - u_{p_j}| + |u_{d_i} - u_{d_j}|\right)$$

où p_i et d_i sont respectivement les noeuds de pickup et dropoff associés à la requête, et u_{p_i} et u_{d_i} les heures de service aux pickup et dropoff de la requête i. Cet opérateur est aussi randomisé comme pour l'opérateur "Worst". Le paramètre contrôlant la randomisation est également p_r.

5.3.2 Etape de Reconstruction

Une fois l'étape de destruction terminée, les requêtes choisies sont enlevées de leurs tournées respectives. La prochaine étape consiste à les réinsérer de la meilleure manière dans la solution, conjointement avec la nouvelle requête R (ligne 6 de l'algorithme 2). Si toutes les requêtes peuvent être réinsérées sans violer aucune contrainte, une nouvelle solution faisable est trouvée. L'algorithme 3 montre le pseudo code associé à cette étape, qui correspond à la ligne 7 de l'algorithme 2. Une liste vide est initialisée au début de la fonction pour stocker les solutions faisables qui vont être trouvées durant le processus de réparation (ligne 2). La liste des opérateurs est ensuite parcourue séquentiellement (L3) pour permettre à chaque opérateur de tenter d'insérer l'ensemble des requêtes dans la solution (L4). Si une solution faisable est trouvée elle est stockée en mémoire (L5). Si des solutions faisables existent, la meilleure au sens de la fonction objectif

(la durée totale des tournées définie en section 3.2.1) est ensuite renvoyée (L7).

Algorithm 3 Fonction ReinsertRequetes

Entrée : solution actuelle s, Liste des requêtes à insérer LR, liste des opérateurs de construction LRO

Sortie : meilleure solution faisable si elle existe

1: **function** REINSERTREQUETES(LR, LRO, s)
2: $L \leftarrow ListeVide()$
3: **for** chaque opérateur o dans LRO **do**
4: $s_{new} \leftarrow InsertRequests(o, LR, s)$
5: **if** $EstFaisable(s_{new})$ **then** $L \leftarrow L \cup s_{new}$
6: **end for**
7: **if** L non vide **then** return $MeilleurSolution(L)$
8: **end function**

Nous avons utilisé deux opérateurs de la littérature, "Deep Greedy" ([Ropke et al., 2007]) et "Regret" ([Diana and Dessouky, 2004]) et avons proposé un opérateur spécifique à notre problème, "Priority" :

Deep Greedy : Exécute la meilleure insertion parmi toutes les insertions faisables de toutes les requêtes encore à insérer. La meilleure insertion est définie comme l'insertion qui provoque la plus faible augmentation de la fonction objectif. La procédure est réitérée jusqu'à ce que plus aucune insertion ne soit faisable ou que toutes les requêtes aient été réinsérées.

Regret operator : Insert la requête avec le regret le plus élevé. La mesure de regret est définie dans [Diana and Dessouky, 2004] et fournit une estimation de la difficulté à insérer la requête plus tard. L'idée est de déterminer pour chaque requête i sa meilleure insertion dans chaque véhicule k, avec le coût d'insertion c_{ik}. Une matrice contenant les coûts d'insertion est ensuite créée, chaque ligne représentant une requête et chaque colonne un véhicule. Si aucune insertion faisable n'existe, une valeur arbitrairement grande est utilisée à la place. Le regret d'une requête i est ensuite calculé comme :

$$\sum_{k} \left(c_{ik} - \min_{j} c_{lj} \right)$$

La requête avec le regret le plus élevé est choisie et insérée dans sa meilleure position. La procédure est réitérée jusqu'à ce que plus aucune insertion ne soit faisable ou que toutes les requêtes aient été réinsérées.

Priority : L'idée de cet opérateur est d'insérer en premier les requêtes à des positions qui auront un faible impact sur l'insertion des autres requêtes. Les étapes suivantes sont répétées jusqu'à ce que le nombre de requêtes à insérer soit atteint où lorsque plus aucune insertion n'est faisable : 1) pour chaque requête i, on calcule (en testant toutes les insertions possibles) le nombre de véhicules dans lesquels il est possible d'insérer i, 2) on sélectionne la requête i avec le plus faible nombre, 3) on calcule pour chaque véhicule (où l'insertion de i est possible) le nombre de requêtes qui peuvent être insérées à l'intérieur, 4) et enfin on sélectionne le véhicule avec le nombre le plus faible.

Les étapes de destruction et de réparation sont les deux étapes fondamentales de l'algorithme HDR. Il est toutefois important de noter que cette heuristique est différente d'une LNS classique. En effet, dès qu'une nouvelle meilleure solution est trouvée, elle n'est pas améliorée mais seulement gardée en mémoire, la recherche repartant systématiquement de la solution d'origine. C'est une contrainte que nous imposons pour des raisons pratiques. En effet, rien n'assure qu'à ce moment du processus de réservation le client va choisir une des propositions qui lui sera faite. Dans le cas où il la validerait, il peut décider de le faire plus tard, 1 minute après par exemple. Durant cet intervalle de temps, il est possible que des requêtes d'autres clients aient été insérées, ce qui peut rendre non faisable la proposition du client. Il est donc nécessaire d'utiliser une procédure de validation à chaque fois qu'un client valide une proposition (cf chapitre 4). Dans le but de rendre cette procédure simple et efficace, le processus de réinsertion doit être aussi simple que possible et perturber le moins possible la solution actuelle.

5.4 Heuristique basée sur Chaînes d'éjection

Dans cette section, nous présentons une heuristique de réinsertion basée sur un tout autre type de voisinage : les chaînes d'éjection. Fred Glover fut le premier à introduire la notion de chaînes d'éjection pour le problème du voyageur de commerce ([Glover, 1992]). Ce genre de voisinages a depuis été utilisé pour d'autres problèmes de tournées de véhicules, comme dans [Rego, 2001] et [Lim and Zhang, 2007]. L'heuristique présentée ici est basée sur le voisinage défini dans [Gendreau et al., 2006]. Nous avons formalisé le problème de réinsertion comme un problème de calcul de plus court chemin contraint sur un graphe orienté construit à partir de la solution courante et nous avons proposé deux méthodes : HG (Heuristique de Graphe) qui permet de trouver une solution grâce à un algorithme de Bellman-Ford modifié et HGA (HG Améliorée) qui est une version récursive de l'heuristique HG permettant d'explorer plus intensément le voisinage.

5.4.1 Modélisation sous forme de graphe

Le problème de réinsertion est modélisé sous forme d'un graphe $G = (V, A)$ où V est l'ensemble des sommets et A l'ensemble des arcs. Nous noterons S la solution actuelle et R une requête non insérée dans cette solution.

Sommets

L'ensemble des sommets V permet de modéliser les différentes requêtes et les véhicules de la solution S. Il est composé de :

- V_R : noeud correspondant à la requête R
- V_1 : l'ensemble des noeuds correspondant aux requêtes actuellement insérées dans la solution S (un noeud pour chaque requête)
- V_2 : l'ensemble des noeuds correspondant à chaque tournée (un noeud pour chaque tournée)

Arcs

L'ensemble des arcs A permet de modéliser les déplacements possibles des différentes requêtes dans les véhicules. Deux types de mouvement sont possibles : l'éjection d'une requête par une

autre et l'insertion d'une requête dans une autre tournée. L'ensemble contient donc deux types d'arcs :

— $A_E : \{a_{ij}, \forall i \in V_1 \cup \{V_R\}, \forall j \in V_1\}$. C'est l'ensemble des arcs dits "d'éjection" qui représentent l'éjection de j hors de sa tournée et l'insertion de i dans l'ancienne tournée de j.

— $A_I : \{a_{ik}, \forall i \in V_1 \cup \{V_R\}, \forall k \in V_2\}$. C'est l'ensemble des arcs dits "d'insertion" qui représentent l'insertion d'une requête i dans la tournée k.

Il n'existe aucun arc reliant 2 tournées entre elles (i.e 2 éléments de V_2).

Valeurs des arcs

A chaque arc défini ci-avant est associée une valeur représentant le coût du mouvement sous-jacent. Le coût est défini comme la différence entre la valeur de la fonction objectif de la solution obtenue une fois le mouvement réalisé et la valeur de l'objectif actuel. L'objectif est la durée totale des tournées (cf section 3.2.1).

Arc d'insertion : Etant donné un arc $a_{ik} \in A_I$, le coût associé c_{ik} est calculé de la manière suivante : si g_i est le gain (positif) obtenu sur la fonction objectif en retirant i de sa tournée et c'_{ik} la différence de coût de la meilleure insertion de i dans k, $c_{ik} = c'_{ik} - g_i$. Lorsque l'insertion de i dans k n'est pas faisable, $c_{ij} = \infty$.

Arc d'éjection : Etant donné un arc $a_{ij} \in A_E$, le coût associé c_{ij} est calculé de la manière suivante : j est tout d'abord enlevé de sa tournée, notée r_j. Si g_i est le gain (positif) obtenu sur la fonction objectif en retirant i de sa tournée et c_{ir_j} la différence de coût engendrée par la meilleure insertion de i dans r_j, alors $c_{ij} = c_{ir_j} - g_i$. Lorsque $i = R$, $g_i = 0$. Lorsque l'insertion de i dans r_j n'est pas faisable, $c_{ij} = \infty$.

Le calcul des coûts nécessite de calculer les insertions des différentes requêtes représentées par $V_1 \cup \{V_R\}$ dans les tournées représentées par l'ensemble V_2.

5.4.2 Plus court chemin contraint

Le problème de la réinsertion est modélisé comme un problème de plus court chemin contraint de V_R jusqu'à n'importe quel noeud $k \in V_2$ où chaque arc du chemin a un coût fini. Un tel chemin est défini par une suite d'arcs provenant de V_R et se finissant dans une tournée k. La figure 5.1 présente un exemple simple avec 3 véhicules. Une couche de sommets est associée à chaque tournée. Chaque sommet de la tournée correspond à une requête servie sur la tournée et un sommet représentant la tournée elle même. Chaque mouvement d'éjection ou d'insertion faisable (avec un coût fini) est représenté par un arc entre les sommets concernés. Les arcs ayant un coût infini (non faisables) ne sont pas représentés.

Le chemin bleu représente le seul chemin menant à une solution faisable dans ce graphe. Chaque arc du chemin correspond à un mouvement d'éjection ou d'insertion qui modifie la solution d'origine S. Appliquer la suite de mouvements définie par le chemin permet de passer d'une solution avec une requête non insérée à une solution où toutes les requêtes sont insérées. Un chemin correspond donc à un mouvement dans le voisinage défini par la structure de graphe. Le coût de chaque arc correspond à la variation de l'objectif lorsque le mouvement d'éjection/insertion associé à l'arc est réalisé et la somme du coût des arcs successifs du chemin représente la valeur de la fonction objectif de la nouvelle solution une fois le mouvement total réalisé. Le plus court chemin correspond donc au mouvement qui permet d'obtenir la solution avec toutes les requêtes insérées ayant le coût le plus faible.

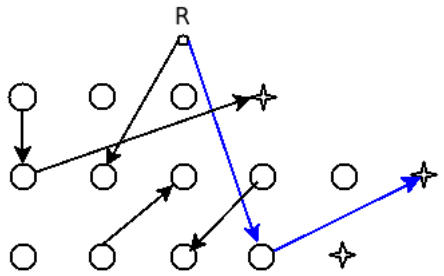


Figure 5.1 – Exemple d'un graphe avec 3 véhicules. R représente la nouvelle requête. Chaque ligne représente les sommets d'une tournée, les ronds représentant les requêtes et la croix étoilée le véhicule.

Il est cependant nécessaire d'ajouter une contrainte pour que le chemin déterminé corresponde à un mouvement faisable : une tournée ne doit être visitée qu'une seule fois dans le chemin. Une tournée est visitée par un chemin lorsque le chemin contient un noeud correspondant à une requête de la tournée ou au noeud de la tournée. En effet, visiter une tournée équivaut à modifier cette dernière, ce qui peut amener à modifier les coûts des arcs entrant ou sortant d'un noeud de cette tournée. Cette contrainte de non-retour permet d'assurer que le mouvement final est faisable et que le coût du chemin correspond au coût final de la solution obtenue.

5.5 Description de l'heuristique HG

La formalisation de graphe exposée section 5.4.1 est au coeur du fonctionnement de l'heuristique HG. L'algorithme 4 présente l'heuristique de réinsertion HG. Il reçoit en entrée la solution actuelle, la requête à réinsérer et TG, un paramètre temporel en minutes définissant l'ensemble des requêtes à considérer pour la recherche. Le graphe défini ci-avant est d'abord crée (L1). Toutes les requêtes de la solution S ne sont pas représentées par des noeuds du graphe. Seules les requêtes dont l'heure de pickup ou de dropoff appartiennent à l'intervalle $[RH - W - TG,\ RH + W + TG]$ sont considérées. Une fois le graphe construit, une heuristique de plus court chemin est lancée pour tenter de déterminer le plus court chemin depuis V_R jusqu'à tous les noeuds du graphe (L2). Le noeud de tournée $k \in V_2$ ayant la plus petite distance (finie) au noeud V_R est ensuite sélectionné (L3). Si aucun n'est trouvé, la recherche est finie en renvoyant un chemin vide. Dans le cas où un noeud est trouvé, le chemin entre V_R et k est déterminé et renvoyé (L5).

Algorithm 4 Heuristique de réinsertion

Entrée : solution actuelle S, requête à réinsérer R, TG
Sortie : meilleur chemin faisable si il existe

1: graphe ← $CreeGraphe$(S, R, TG)
2: dist_dict, chemin_dict ← $BFSModifie$(graphe, R)
3: meilleur_tournee ← $RecupereMeilleurTournee$(dist_dict)
4: **if** meilleur_tournee existe **then**
5: meilleur_chemin ← $RecupereMeilleurChemin$(chemin_dict)
return meilleur_chemin
6: **end if**

Les deux sous-sections suivantes décrivent avec plus de détails l'heuristique de plus court

chemin utilisée et la reconstruction du plus court chemin.

5.5.1 Heuristique de plus court chemin

Le problème de plus court chemin contraint rencontré dans la réinsertion est NP-Complet ([Gendreau et al., 2006]). Nous nous tournons donc vers une heuristique, une résolution exacte étant hors de portée pour une application temps réel sur nos instances. Nous avons ainsi choisi d'adapter l'algorithme de Bellman-Ford [1] pour résoudre ce problème. L'algorithme 5 présente les détails de l'heuristique.

Algorithm 5 Bellman-Ford modifié

Entrée : ensemble des sommets V, ensemble des arcs A, sommet de départ V_R

Sortie : Plus courtes distances pour chaque noeud *distance*, prédécesseur de chaque noeud *pred*

```
1:  function BFSModifie(V, A, V_R)
2:      for chaque sommet v dans V do
3:          pred[v] ← null
4:          distance[v] ← infini
5:      end for
6:      pred[V_R] ← V_R
7:      distance[V_R] ← 0
8:      deja_vu← ListeVide() ;
9:      for 1 ≤ i ≤ Taille(V) − 1 do
10:         for chaque arête (u, v) dans A do
11:             if distance[u] < distance[v] + c_uv then
12:                 liste_veh ← TrouveListeTourneeDansChemin(u, pred)
13:                 if Vehicule(v) pas dans liste_veh et v pas dans deja_vu then
14:                     distance[v] ← distance[u] + c_uv
15:                     pred[v] ← u
16:                     deja_vu.AjoutFin(v)
17:                 end if
18:             end if
19:         end for
20:     end for
        return distance, pred
21: end function
```

Les variables "pred" et "distance" sont des dictionnaires indiquant respectivement le prédécesseur de chaque noeud dans le plus court chemin actuel à ce noeud et la taille du plus court chemin actuel connu pour chaque noeud. Ces dictionnaires sont mis à jour régulièrement durant la recherche. "pred" est initialisé à une valeur vide pour chaque noeud sauf V_R, qui est son propre prédécesseur. "distance" est initialisé à infini pour tous les noeuds sauf V_R qui est à une distance 0 de lui-même (lignes 2 à 7). Une liste des sommets déjà rencontrés dans la recherche est mise à jour durant la recherche, initialisée à une valeur vide (L8). Une fois les variables initialisées, l'heuristique effectue autant d'itérations que le nombre de noeuds moins 1 (L9). A chaque itération, l'ensemble des arcs est parcouru comme pour l'algorithme de Bellman-Ford classique (L10). Un test est réalisé pour chaque arc pour déterminer si les informations sur les

1. `https://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford`

chemins doivent être mises à jour : pour chaque arc (u, v), si un chemin plus court vers v est possible en passant par u (L11), la liste des tournées visitées par le plus court chemin menant à u est calculée (L12). Si la tournée associée au noeud v n'est pas dans la liste des tournées et si le noeud v n'a pas encore été vu dans l'heuristique (L13), les informations concernant le plus court chemin sont mises à jour (lignes 14 et 15). La liste des sommets vus par l'heuristique est ensuite mise à jour (L16). Une fois les itérations terminées, les dictionnaires "pred" et "distance" sont renvoyés en sortie de l'algorithme (L16).

Les parties du pseudo-code en gras représentent les mises à jour que nous avons apportées pour adapter l'algorithme Bellman-Ford à notre problème. Deux ajouts ont été réalisés :

1. A chaque itération, la liste des véhicules pour le sommet u est déterminée à partir du dictionnaire des prédécesseurs *pred* (L12). Si le véhicule associé à v se trouve dans cette liste, alors la paire (u, v) n'est pas éligible (L13).
2. La vérification précédente n'est pas suffisante pour assurer que la contrainte sur le "non retour" du chemin est vérifiée. En effet, supposons r_v ne soit pas contenu dans les véhicules visités sur le chemin de u. v est donc accepté comme successeur de u. Soit j un prédécesseur de u. Il est possible que plus tard dans le déroulement de l'algorithme, le plus court chemin menant à j change et que le chemin contienne désormais le véhicule r_v. La contrainte serait ainsi violée. Pour pallier cette faille sans pour autant ajouter trop de complexité à l'algorithme, nous ajoutons une règle simple : une fois qu'un sommet a été parcouru, le plus court chemin associé ne peut plus être modifié. Ceci est implémenté en maintenant une liste des sommets visités (L8 et L16). Une simple vérification que le sommet est présent dans la liste est ainsi réalisée à chaque itération (L13). Bien que cette règle semble restrictive, elle semble en pratique donner de bons résultats tout en gardant l'heuristique simple et rapide.

5.5.2 Récupération du meilleur chemin

L'algorithme de plus court chemin présenté section 5.5.1 ne renvoie pas directement le mouvement à réaliser pour trouver la solution mais renvoie deux dictionnaires : le dictionnaire des distances et le dictionnaire des prédécesseurs. Pour déterminer le plus court chemin, il faut dans un premier temps identifier le noeud k représentant une tournée ($k \in V_2$) dont la valeur associée dans le dictionnaire des distances est finie et minimale. Si ce noeud n'existe pas, cela implique qu'aucun mouvement faisable n'est trouvé par la réinsertion. Lorsqu'un tel noeud existe, le chemin inverse entre V_R et k est reconstruit en parcourant le dictionnaire des prédécesseurs de la manière suivante : le chemin est initialisé comme une liste vide à laquelle k est ajouté en première position. Le prédécesseur k_1 de k est ensuite lu dans le dictionnaire et ajouté au chemin, à la suite de k. Le prédécesseur k_2 de k_1 est ensuite lu et ajouté à la liste et ainsi de suite jusqu'à arriver à V_R. Le plus court chemin est trouvé en parcourant dans l'ordre inverse la liste ainsi obtenue.

5.6 Description de l'heuristique HGA

Dans le cas où aucun chemin faisable n'est trouvé par HG, la recherche s'arrête et ce même si le temps imparti n'est pas écoulé. Pour pallier cet inconvénient, nous avons proposé une version améliorée que nous appellerons HGA. L'algorithme 6 présente le pseudo-code associé à

HGA. Les paramètres d'entrée sont les mêmes que pour HG en ajoutant $GTAB$ qui est expliqué section 5.6.1. La recherche commence par la création du graphe (L1) défini en section 5.4.1. Une liste des requêtes déjà ré-insérées est maintenue tout au long de la recherche et initialisée avec R (L2). La fonction principale récursive de l'algorithme est ensuite appelée (L3) et le plus court chemin est ensuite renvoyé si il est trouvé (L4).

Algorithm 6 Heuristique de réinsertion HGA

Entrée : solution actuelle S, requête à réinsérer R, TG, $GTAB$
Sortie : meilleur chemin faisable si il existe

1: graphe ← *CreeGraphe*(S, R, TG)
2: req_deja_ins ← *Liste*(R)
3: meilleur_sommet, meilleur_chemin ← *RecursiveReinsertion*(R, graphe, ListeVide(), req_deja_ins, TG, GTAB)
4: **if** meilleur_sommet existe **then**
    return meilleur_sommet, meilleur_chemin
5: **end if**

L'algorithme 7 donne le pseudo-code associé à la fonction RecursiveReinsertion. C'est une fonction récursive qui est appelée jusqu'à l'obtention d'un mouvement final où jusqu'à ce que le temps imparti soit achevé.

Algorithm 7 Fonction principale de HGA

1: **function** RecursiveReinsertion(requete, graphe, chemin_act, req_deja_ins, TG, GTAB)
2: dist_dict, chemin_dict ← *BFSModifie*(graphe, requete)
3: **if** *NombreCheminFaisable*(dist_dict) = 0 **then**
4: nouv_req ← *ExecuteMeilleurCheminPartiel*(dist_dict, req_deja_ins, graphe, GTAB)
5: **if** nouv_req n'existe pas **then**
 return
6: **end if**
7: req_deja_ins.*AjoutFinDeListe*(nouv_req)
8: chemin_a_realiser ← chemin_dict[nouv_req]
9: chemin_act ← *MetAJourChemin*(chemin_act, chemin_a_realiser)
10: liste_tour_impactes ← *TrouveListeTourneeDansChemin*(chemin_a_realiser)
11: graphe ← *EnleveArcsObsoletes*(graphe, nouv_req, list_tour_impactes)
12: graphe ← *MetAJourGraphe*(requete, graphe, list_tour, TG)
 return *RecursiveReinsertion*(nouv_req, graphe, chemin_act, req_deja_ins)
13: **end if**
14: meilleur_tournee ← *RecupereMeilleurTournee*(dist_dict)
15: meilleur_chemin ← *RecupereMeilleurChemin*(chemin_dict)
16: meilleur_chemin ← *MetAJourChemin*(chemin_act, meilleur_chemin)
 return meilleur_chemin
17: **end function**

L'idée principale de cette fonction est la suivante : lorsque une recherche du plus court chemin n'aboutit pas à un chemin faisable de V_R jusqu'à $k \in V_2$, un chemin allant de V_R à $i \in V_1$ est appliqué sur la solution d'origine. On obtient donc une nouvelle solution avec V_R inséré et i éjecté. La procédure est relancée avec i à la place de V_R. La procédure est lancée

autant de fois que nécessaire jusqu'à expiration du temps ou jusqu'à obtenir une solution avec toutes les requêtes insérées. Les requêtes déjà éjectées sont taboues pendant un certain nombre d'itérations et ne peuvent plus être choisies pour être éjectées durant cette période. Les paramètres d'entrée de la fonction sont les suivants : "requete" correspond à la requête actuellement éjectée, "graphe" correspond au graphe de la solution actuelle, "chemin_act" représente le chemin actuel et permet de stocker tous les chemins successifs réalisés par les appels récursifs de la fonction pour être en mesure de renvoyer le chemin complet à la fin de l'algorithme, "req_deja_ins" représente les requêtes qui ont été éjectées aux itérations précédentes et TG est un paramètre ayant le même rôle que dans l'algorithme 2.

La fonction commence par lancer une recherche du plus court chemin sur le graphe courant avec l'heuristique décrite dans l'algorithme 11 (L2). Si aucun chemin allant de V_R jusqu'à un noeud $k \in V_2$ n'a été trouvé (L3), un chemin partiel va être exécuté (L4). Un chemin "partiel" est un chemin partant du noeud "requete" jusqu'à un noeud quelconque "nouv_req" $\in V_1$ aboutissant à l'insertion de "requete" dans la solution et à l'éjection de "nouv_req". Si aucun chemin n'est trouvé (L5), la recherche est finie. La nouvelle requête éjectée est ajoutée à la liste des requêtes qui ont déjà été éjectées (L7). Le chemin actuel est ensuite mis à jour (lignes 8 et 9) et la liste des tournées impactées par ce chemin est ensuite déterminée. Tous les arcs du graphe reliés à au moins un noeud appartenant aux tournées impactées sont potentiellement obsolètes. Ces arcs sont donc enlevés du graphe (L11), ce dernier étant ensuite mis à jour en recalculant les arcs manquant avec les nouvelles informations (L12). Cette étape permet de ne pas à avoir à recalculer le graphe en entier si seulement 1 ou 2 tournées ont été impactées. La fin de la fonction (lignes 14 à 16) a pour but de retrouver le meilleur chemin final.

5.6.1 Chemin partiel

La ligne 4 de l'algorithme 7 consiste à exécuter un chemin partiel dans le but d'insérer la requête actuelle et d'en éjecter une autre. Lorsqu'une requête est éjectée, elle ne peut plus être éjectée pendant les $GTAB$ prochaines itérations de l'heuristique (i.e taboue pendant $GTAB$ itération). Nous utiliserons toujours une valeur de $GTAB > 1$ pour éviter une possible récursion infinie. Le chemin partiel qui sera exécuté est déterminé grâce au dictionnaire des distances passé en paramètre de la fonction. Nous utiliserons trois objectifs différents pour définir le meilleur chemin partiel :

PCPA : Le chemin menant à la requête qui a le plus d'arêtes sortantes, en espérant ainsi laisser une plus grande marge de manoeuvre à la suite de l'algorithme.
PCPB : Le chemin le plus court parmi tous les chemins partiels faisables.
PCPC : Le chemin menant à la requête qui a le plus d'arêtes sortantes. Si plusieurs chemins sont équivalents, le chemin le plus court est choisi.

La liste des requêtes est triée en fonction de l'objectif choisi, la première requête non taboue de la liste étant sélectionnée. Une comparaison des trois objectifs est proposée en section 5.7.2. Il est possible qu'aucun chemin partiel ne soit trouvé (ligne 5) et ce pour deux raisons : aucun chemin partiel n'existe ou alors les chemins existants mènent à des requêtes qui ne peuvent plus être enlevées.

5.7 Expérimentations

L'objectif de cette section est d'évaluer l'impact des différentes heuristiques proposées et le comportement des paramètres libres associés. Une simulation de référence est lancée sur chaque instance avec l'algorithme online pour avoir un résultat de référence. Le paramètre MNN déterminant le nombre de noeuds de pickup et dropoff considérés dans l'optimisation (cf section 4.3.4) est fixé à 1 pour mieux évaluer l'apport des heuristiques de réinsertion. Une valeur de référence est également calculée pour l'heuristique de réinsertion testée. Cette valeur est obtenue avec une seule simulation pour une heuristique déterministe (HG) et en utilisant la moyenne de 10 simulations lorsque l'heuristique possède un aspect stochastique (HDR et HGA). HGA est considérée comme stochastique car aucune règle précise n'est donnée lorsque plusieurs chemins partiels sont égaux suivant la métrique utilisée par la politique de choix (cf 5.6.1). En particulier, la fonction de tri implémentée par python a un comportement qui n'est pas entièrement déterministe lorsque plusieurs chemins sont égaux. Bien que cette variation soit très faible, nous choisissons de lancer 10 simulations pour obtenir des résultats robustes.

Si RV est le résultat obtenu par l'algorithme online et V le résultat obtenu par la réinsertion pour une métrique donnée, nous calculons pour comparaison le pourcentage d'amélioration relative $\frac{V-RV}{RV} * 100$.

Toutes les simulations sont menées sur une machine contenant 32GB de RAM et un processeur quadri-core Intel Xeon E3-1245 cadencé à 3.40GHZ. Le module est implémenté entièrement en Python/Cython [2].

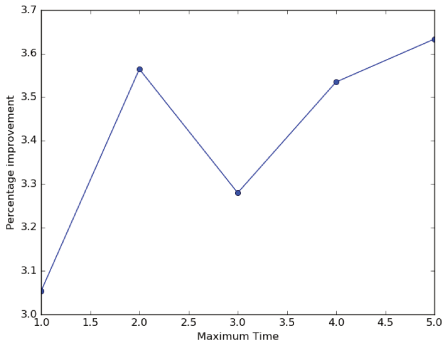
5.7.1 Expérimentation sur l'algorithme HDR

Nous commençons les expérimentations en testant l'impact des paramètres importants, T et MT définis dans la section 5.3. Concernant les valeurs des autres paramètres libres, des expérimentations préliminaires par dichotomie nous ont conduit à $k_{min} = 3$, $k_{max} = 10$. Nous avons repris une valeur de la littérature $p_r = 4$ ([Vallée et al., 2017]).

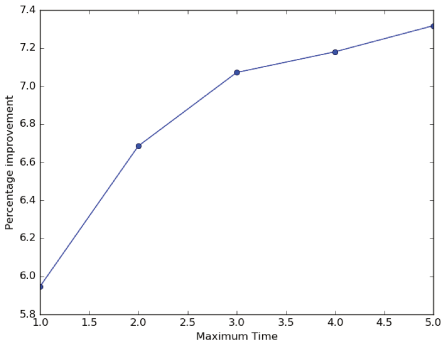
Etude de l'impact du temps maximum de recherche MT

Nous chercherons à évaluer ici l'impact du temps de recherche de HDR (MT) sur le pourcentage total de requêtes servies. La valeur de T est fixée à 15 minutes pour les deux groupes. La figure 5.2 montre les résultats sur les deux groupes d'instances. On peut observer dans les deux groupes une différence significative entre $MT = 1$ et $MT = 2$ secondes. Cela indique que laisser au moins 2 secondes à HDR est très bénéfique pour les performances. Au delà de 2 secondes il n'y a plus d'améliorations significatives pour le groupe A, la faible variation étant probablement due à la nature stochastique de l'algorithme. Pour le groupe B, on peut constater une amélioration jusqu'à 5 secondes, l'amélioration la plus importante étant provoquée par le passage de 1 à 3 secondes. Le groupe B bénéficie plus de l'augmentation du temps maximal, probablement pour les raisons expliquées en section 5.7.1. Nous choisissons donc de limiter le temps maximum à 3 secondes, considérant que ce temps est acceptable pour l'utilisateur et qu'il n'implique pas de détérioration sur la qualité de service et sur les performances de HDR. Une valeur de 2 secondes serait aussi acceptable.

2. https ://cython.org/



(a) Groupe A



(b) Groupe B

Figure 5.2 – Amélioration relative selon le temps maximal MT accordé à la réinsertion pour chaque groupe d'instance. L'axe des y représente la moyenne de l'amélioration relative sur toutes les instances du groupe et l'axe des x les différentes valeurs de MT (secondes) : $1, 2, 3, 4, 5$

Etude de l'impact de T

Nous évaluerons ici l'impact du paramètre T sur le pourcentage total de requêtes servies. Le temps maximum de HDR (MT) est limité à 3 secondes, selon ce qui a été déterminé à la section précédente. La figure 5.3 montre les résultats des simulations pour chaque groupe.

On observe dans les deux groupes une augmentation de la performance jusqu'à une certaine valeur qui est de 60 minutes pour le groupe A et de 15 minutes pour le groupe B. Ces valeurs offrent un plus grand voisinage que $T = 0$, ce qui permet à l'algorithme HDR d'accéder à un plus grand nombre de requêtes pertinentes. La seule exception notable est la baisse soudaine de performance pour $T = 45$ minutes qui est difficilement explicable. Le comportement au delà de ces valeurs est ensuite différent pour les deux groupes. Dans le groupe B, la performance diminue au fur et à mesure de l'augmentation de T jusqu'à devenir moins bonne que $T = 0$ alors que dans le groupe A la performance se stabilise autour d'une valeur moyenne comparable avec les performances de $T = 60$ minutes et avec des meilleures performances que $T = 0$. Le tableau 5.1 nous permet d'expliquer ce comportement. *Voisinage* représente le nombre moyen de requêtes et *Faisable* le pourcentage de mouvements faisables obtenus en exécutant une simulation sur chaque instance. On observe que la taille des voisinages est en moyenne proche pour les deux groupes. Cependant, il apparaît clairement qu'il est plus difficile de trouver des mouvements faisables dans le groupe A, ce qui confirme la difficulté de ces instances (cf 3.3.2). En effet, les requêtes clients du groupe B suivent des pattern géographiques et temporels plus régulier que celles du groupe A, rendant la recherche de mouvements faisables plus facile dans le groupe B. Le tableau 5.1 nous apprend également que le pourcentage de mouvements faisables diminue de manière beaucoup plus significative dans le groupe B lorsque T augmente. C'est l'explication principale du comportement observé : lorsque T augmente, le pourcentage de mouvements faisables reste relativement similaire et suffisant dans le groupe A alors que ce n'est pas le cas dans le groupe B. Il est donc important, particulièrement pour le groupe B, de restreindre l'ensemble des requêtes considérées. Ne pas utiliser de filtre (i.e mettre T à une valeur infinie) serait très préjudiciable pour les performances.

Pour la suite des expérimentations, nous retiendrons pour le paramètre T la valeur de 60 minutes pour le groupe A et de 15 minutes pour le groupe B. Utiliser une valeur différente en fonction du groupe d'instance permet à Padam d'adapter au mieux ses algorithmes en fonction du territoire sur lequel elle opère, les deux territoires étant complètement indépendants.

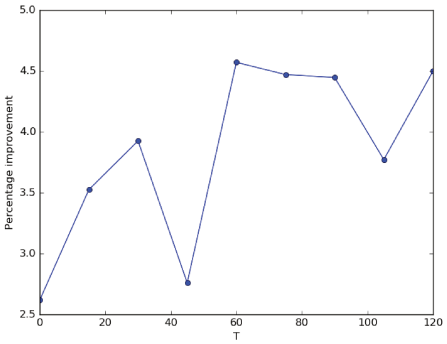
T	0	60	120
Voisinage	23.71	55.78	62.81
Faisable	1.72	1.68	1.43

(a) Groupe A

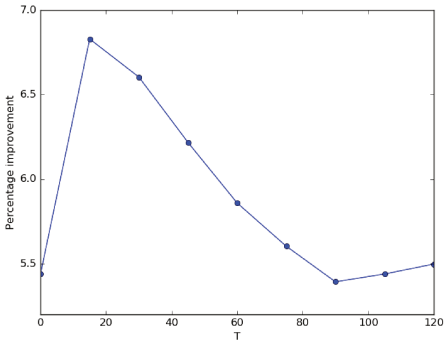
T	0	60	120
Voisinage	13.51	46.29	66.20
Faisable	13.49	8.23	6.12

(b) Groupe B

Tableau 5.1 – Nombre moyen de requêtes et de mouvements faisables selon la valeur de T (minutes) pour chaque groupe



(a) Groupe A



(b) Groupe B

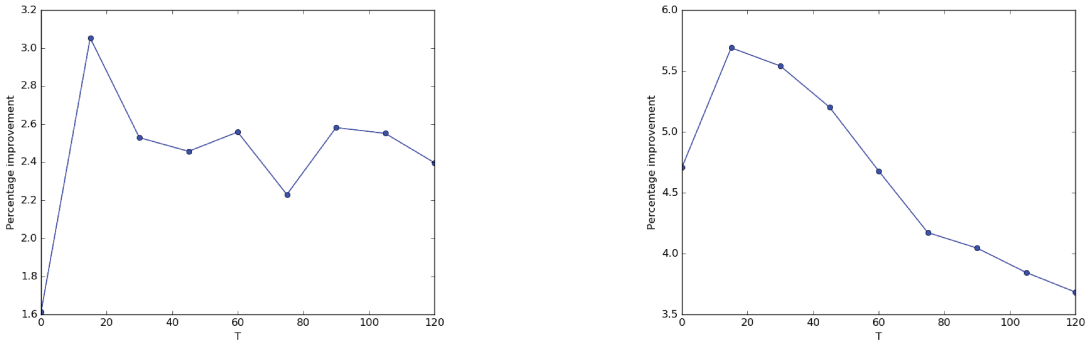
Figure 5.3 – Amélioration relative selon la valeur de T pour chaque groupe d'instance avec $MT = 3$ secondes. L'axe des y représente la moyenne de l'amélioration relative sur toutes les instances du groupe et l'axe des x les différentes valeurs de T (minutes), de 0 à 120 par pas de 15

Les expérimentations précédentes ont été faites avec $MT = 3$ secondes. Nous avons également observé le comportement en fonction de T lorsque $MT = 1$ seconde, présenté en figure 5.4. Le comportement sur le groupe B est très similaire au comportement observé lorsque $MT = 3$ secondes. En revanche, le comportement du groupe A est très différent. La meilleure valeur n'est plus 60 minutes mais 15 minutes et les valeurs supérieures à 15 minutes présentent de moins bons résultats comparés à 15 minutes. Cela s'explique par le fait que pour les valeurs supérieures à 15 minutes, 1 seconde n'est pas suffisante pour explorer les mouvements pertinents du voisinage. Ce comportement n'est plus observé lorsque $MT = 3$ secondes car l'augmentation du temps permet d'explorer suffisamment les mouvements pertinents du voisinage. Ceci nous montre qu'il est très important de considérer l'impact conjoint de MT lors du choix de T. En effet, la figure 5.4 nous aurait conduit à choisir $T = 15$ minutes, ce qui produit des résultats significativement moins bons que $T = 60$ minutes lorsque $MT = 3$ secondes qui est la valeur finale utilisée pour MT (cf figure 5.3).

Evaluation de l'impact de l'heuristique de réinsertion HDR

L'objectif des expérimentations de cette section est d'évaluer le bénéfice apporté par HDR en comparaison avec l'algorithme online simple. La principale métrique d'intérêt sera le pourcentage total de requêtes servies. Nous montrerons également la durée totale des véhicules, qui est l'objectif principal minimisé par les algorithmes du service de Padam. Les valeurs de T et MT utilisées sont celles déterminées aux sections précédentes.

Le tableau 5.2 présente les résultats du groupe A avec 12, 13 et 14 véhicules pour chaque



(a) Groupe A (b) Groupe B

Figure 5.4 – Amélioration relative selon la valeur de T pour chaque groupe d'instance avec $MT = 1$ seconde. L'axe des y représente la moyenne de l'amélioration relative sur toutes les instances du groupe et l'axe des x les différentes valeurs de T (minutes), de 0 à 120 par pas de 15

instance. O représente le pourcentage de requêtes servies avec l'algorithme online, $O{+}R$ est le pourcentage de requêtes servies lorsque HDR est utilisé, *Imp* est l'amélioration relative de HDR sur l'algorithme online en terme de requêtes servies et *Imp-D* l'amélioration relative de HDR sur l'algorithme online en terme de durée des tournées.

Le tableau rappelle que les instances du groupe A sont très difficiles, car l'algorithme online ne permet de servir qu'une partie de l'ensemble des requêtes, parfois moins du tiers (cf chapitre 4). Cependant, HDR permet d'améliorer presque toutes les instances, avec une amélioration relative de 4.57% (respectivement 4.89% et 5.58%) avec 13 véhicules (avec 12 et 14 véhicules respectivement). Les tournées obtenues par l'algorithme HDR ont en moyenne une durée similaire comparées aux tournées obtenues par l'algorithme online, avec une légère amélioration de ce temps sur plusieurs instances pour l'algorithme HDR. HDR permet donc de servir plus de requêtes sans augmenter les coûts de fonctionnement. En comparant les résultats obtenus avec 12 et 13 véhicules, nous observons que l'algorithme online sert 45.82% des requêtes avec 12 véhicules comparé à 49.68% avec 13 véhicules, tandis que HDR sert 48.25% avec 12 véhicules. Cela signifie qu'en dépit de la difficulté des instances, HDR permet de combler plus de 75% du déficit causé par le retrait d'un véhicule, le comblant même totalement dans le cas des instances A_120 et A_177. Les résultats montrent aussi qu'avec 13 véhicules, HDR permet d'insérer en moyenne plus de requêtes qu'avec l'algorithme online avec 14 véhicules, faisant mieux sur la moitié des instances. Cela signifie que HDR permet de gagner 1 véhicule sur ces 5 instances, montrant ainsi l'intérêt bénéfique qu'elle apporte même sur des instances très difficiles.
L'analyse des résultats obtenus en fonction de la variation du nombre de véhicules montre que l'amélioration de l'algorithme online est plus faible de 13 à 14 véhicules, le pourcentage de requêtes servies n'étant augmenté que de 1.9%. En revanche, HDR fournit une amélioration encore plus élevée avec 14 véhicules. Ceci peut-être expliqué par le fait que l'algorithme online n'exploite pas pleinement le potentiel offert par le nouveau véhicule à cause de son comportement "myope" tandis que HDR couvre un voisinage plus large, augmentant ainsi les chances d'exploiter au mieux le nouveau véhicule.
On observe dans des rares cas que HDR implique une détérioration en terme de pourcentage de requêtes servies. Ceci peut-être expliqué par le fait que le processus de réinsertion permet d'accepter des requêtes difficiles qui sont rejetées lorsqu'elle n'est pas activée, impliquant une

dégradation des véhicules et rendant plus difficile l'insertion des futures requêtes. Ceci semble être confirmé sur certaines instances comme A_216 et A_288b, où l'on remarque que HDR sert moins de requêtes et obtient une durée des tournées plus élevée ou similaires à l'online.

Ve	12				13				14			
INS	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD
A_120	67.5	74.33	**10.12**	**-0.40**	74.17	82.00	**10.56**	**-1.40**	75.83	83.5	**10.11**	0.74
A_145	53.10	55.66	**4.81**	1.06	57.24	60.76	**6.14**	**-0.73**	62.06	61.52	-0.89	**-2.91**
A_159	60.38	62.77	**3.96**	3.21	66.04	68.49	**3.71**	2.57	65.41	71.95	**10.00**	2.04
A_177	42.37	46.67	**10.13**	4.88	44.63	46.87	**4.56**	**-0.86**	46.33	47.23	**1.95**	**-2.55**
A_216	43.52	42.78	-1.70	1.04	44.44	45.09	**1.46**	**-0.43**	45.37	48.29	**6.43**	0.38
A_233	51.07	52.58	**2.94**	0.62	55.79	58.67	**5.15**	3.57	60.94	61.46	**0.85**	**-0.38**
A_271	34.32	34.98	**1.93**	**-0.21**	36.53	39.34	**7.68**	**-3.90**	38.75	41.92	**8.19**	**-0.91**
A_288	34.03	34.06	**0.10**	0.88	34.72	35.49	**2.20**	0.39	35.07	37.92	**8.12**	0.95
A_288b	45.49	51.18	**12.52**	**-0.58**	54.86	53.96	-1.65	**-0.14**	55.55	57.53	**3.56**	**-0.89**
A_360	26.39	27.47	**4.11**	0.30	28.33	30.00	**5.88**	**-2.86**	30.27	32.56	**7.52**	0.48
Avg.	45.82	48.25	**4.89**	1.08	49.68	52.05	**4.57**	**-0.38**	51.56	54.39	**5.58**	**-0.31**

Tableau 5.2 – Performance de HDR sur les instances du groupe A. Chaque instance est nommée G_N où G est le nom du groupe (A) et N le nombre de requêtes.

Le tableau 5.3 expose les résultats pour le groupe B. Ces résultats sont plus homogènes que pour le groupe A, l'algorithme HDR apportant une amélioration sur chaque instance. Avec 6 véhicules, l'algorithme HDR donne une amélioration moyenne de 7.07% avec une durée totale des tournées similaire à celle obtenue par l'algorithme online, ce qui est une amélioration significative. Avec 5 véhicules, elle permet une amélioration de 11.16% pour atteindre en moyenne 81.46% de requêtes servies. Ce pourcentage moyen est plus grand que le pourcentage moyen atteint par l'algorithme online et 6 véhicules. C'est en particulier vérifié sur deux tiers des instances (6). Cela signifie que l'algorithme HDR permet en moyenne d'utiliser un véhicule de moins sur les 6, ce qui est un gain considérable pour le transporteur. Il en va de même lorsqu'on compare les résultats avec 6 et 7 véhicules : l'algorithme HDR avec 6 véhicules permet de servir plus de requêtes que l'algorithme online seul avec 7 véhicules sur **toutes** les instances. HDR permet donc au transporteur de garder 6 véhicules tout en ayant de meilleures performances qu'avec 7 véhicules, le tout avec un coût similaire. C'est également un gain considérable.
Nous observons également que lorsque le nombre de véhicules est fixé, l'algorithme HDR tend à avoir de meilleures performances lorsque le nombre de requêtes augmente. Cela peut potentiellement être expliqué par le fait que plus le nombre de requêtes est élevé, plus l'algorithme online est loin de l'arrangement optimal pour ces requêtes. Une heuristique moins "myope" comme l'algorithme HDR a ainsi plus de possibilités pour optimiser la solution, amenant au final un plus grand pourcentage de requêtes à être servies sur les instances du groupe B.

Synthèse des expérimentations sur l'algorithme HDR

Pour les deux groupes, il est important de laisser un temps de recherche MT d'au moins 2 secondes, l'impact de MT sur le groupe B étant visible jusqu'à 5 secondes. La valeur finale choisie de 3 secondes permet à l'algorithme HDR de pouvoir être utilisé en temps réel tout en maintenant un temps de réponse faible et donc une qualité de service élevée. Les expérimentations indiquent que le paramètre T a une très grande importance, particulièrement sur le groupe B, et que sa valeur doit être déterminée en fonction de la valeur de MT.

Ve	5				6				7			
INS	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD
B_200	84.5	88.95	**5.27**	5.27	90.5	92.75	**2.49**	**-2.42**	89.5	92.0	**2.79**	1.93
B_301	82.06	91.06	**10.97**	3.95	89.04	92.29	**3.66**	**-1.61**	91.03	94.15	**3.43**	**-0.80**
B_398	86.93	90.70	**4.34**	1.29	88.94	91.21	**2.54**	0.39	87.44	91.88	**5.08**	1.99
B_498	76.51	85.94	**12.34**	2.42	85.94	90.84	**5.70**	2.40	87.35	93.88	**7.47**	0.97
B_607	72.32	82.19	**13.64**	0.85	80.72	88.37	**9.47**	1.88	84.51	91.57	**8.34**	**-0.38**
B_709	69.25	78.22	**12.95**	0.87	80.82	84.79	**4.92**	0.96	84.34	89.01	**5.53**	1.32
B_802	67.83	75.01	**10.58**	**-0.29**	72.57	82.77	**14.05**	2.08	78.05	86.66	**11.02**	0.06
B_909	61.45	75.01	**17.29**	0.48	73.68	81.38	**10.45**	1.17	76.98	85.45	**11.00**	2.90
B_1011	61.03	69.00	**13.06**	**-0.02**	70.23	77.51	**10.37**	0.44	75.57	82.11	**8.65**	0.38
Avg.	73.54	81.46	**11.16**	1.65	81.38	86.88	**7.07**	0.59	83.86	89.64	**7.04**	0.93

Tableau 5.3 – Performance de HDR sur les instances du groupe B. Chaque instance est nommée G_N où G est le nom du groupe (B) et N le nombre de requêtes

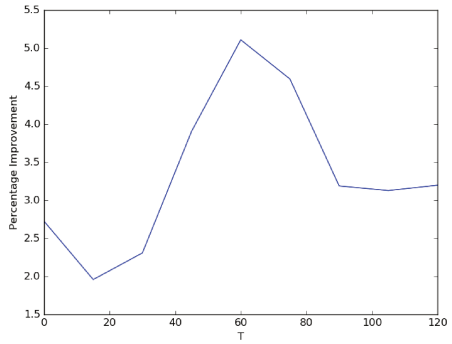
Les résultats finaux indiquent que l'algorithme HDR est capable de servir plus de requêtes sur presque toutes les instances avec des coûts de fonctionnement similaires à l'online. Ils indiquent aussi que l'algorithme HDR exploite plus pleinement le potentiel offert par l'ajout d'un nouveau véhicule. L'apport principal de l'algorithme HDR est de pouvoir compenser le retrait d'un véhicule sur de nombreuses instances, ce qui apporte un gain significatif aux transporteurs, et ce malgré la difficulté et la grande taille des instances. On peut donc conclure que l'algorithme HDR est utilisable en temps réel dans le système de PADAM avec une garantie d'efficacité.

5.7.2 Expérimentations sur les heuristiques HG et HGA

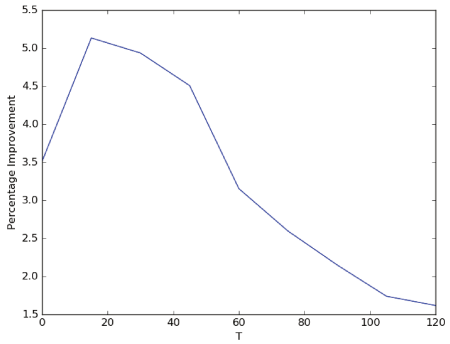
Etude de l'impact de TG

L'objectif est d'évaluer l'impact de TG sur HGA. Rappelons que TG détermine la taille de l'intervalle où sont choisies les requêtes candidates pour être déplacées de leur position actuelle. Nous avons lancé des simulations avec différentes valeurs de TG avec $GTAB$ fixé à 10000 (i.e requête taboue pour toujours) et $PCPA$ pour le choix d'un meilleur chemin partiel. La figure 5.5 expose les résultats sur les deux groupes d'instances. On remarque clairement la même tendance sur les deux groupes : la performance augmente lorsque TG augmente jusqu'à atteindre un maximum puis redescend au fur et à mesure que TG continue à augmenter. Ce comportement est attendu et similaire à celui observé sur le groupe B dans la section 5.7.1. On remarque de plus que l'impact de TG est très grand car les performances peuvent varier du simple au triple en fonction de la valeur prise par le paramètre. Ceci est du au fait que le calcul du graphe évolue de manière polynomiale avec le nombre de requêtes et devient trop long si le nombre de requêtes est trop élevé.

La meilleure valeur pour A est 60 minutes tandis que la meilleure valeur pour B est 15 minutes. Nous utiliserons ces valeurs de TG pour la suite des expérimentations.



(a) Groupe A

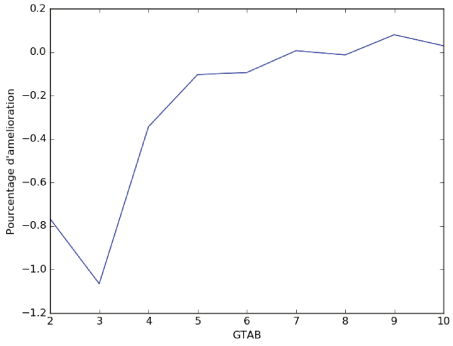


(b) Groupe B

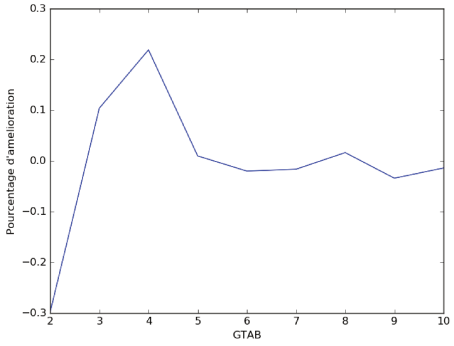
Figure 5.5 – Amélioration relative selon la valeur de TG pour chaque groupe d'instance. L'axe des y représente la moyenne de l'amélioration relative sur toutes les instances du groupe et l'axe des x les différentes valeurs de TG (minutes) : 0, 15, 30, 45, 60, 75, 90, 105 et 120

Etude de l'impact de $GTAB$

Nous évaluerons ici l'impact du paramètre $GTAB$, qui empêche les requêtes d'être enlevées pendant un certain nombre d'itérations. La figure 5.6 expose les résultats obtenus sur les deux groupes d'instances, en comparant chaque valeur de $GTAB$ testée à une valeur arbitrairement grande (10000) qui équivaut à rendre une requête taboue pour toute la recherche. La politique de choix du meilleur chemin partiel est $PCPA$.
On observe dans les deux groupes que le paramètre a presque toujours un impact négatif et dans les rares cas où l'impact est positif, il est très proche de 0 (0.25% au mieux). Les tendances affichées par les deux courbes sont difficiles à interpréter mais semblent indiquer qu'une grande valeur pour le paramètre est meilleure. La principale leçon des ces expérimentations est que la politique de tabou ne procure pas d'avantage significatif à HGA. Nous choisissons donc de désactiver cette fonctionnalité ($GTAB = 10000$) pour la suite de nos expérimentations, le faible apport ne justifiant pas d'ajouter un paramètre de plus à l'heuristique.



(a) Groupe A



(b) Groupe B

Figure 5.6 – Amélioration relative selon la valeur de $GTAB$ pour chaque groupe d'instance par rapport à la valeur $GTAB = 10000$. L'axe des y représente la moyenne de l'amélioration relative sur toutes les instances du groupe et l'axe des x les différentes valeurs de $GTAB$ allant de 2 à 10

Etude de l'impact du critère de choix du meilleur chemin

Nous comparerons ici les différentes politiques de choix du meilleur chemin : $PCPA$, $PCPB$ et $PCPC$. Le tableau 5.4 montre les résultats obtenus avec les trois politiques sur les deux groupes d'instances. Sur le groupe A, $PCPA$ permet d'apporter une amélioration relative de plus de 20% sur $PCPC$ et de plus de 10% sur $PCPB$. L'apport est moins élevé sur le groupe B où l'écart entre les différentes politiques est plus faible. Le constat est néanmoins le même sur les deux groupes : $PCPA$ est la politique qui donne les meilleurs résultats, suivi par $PCPB$ et $PCPC$, $PCPA$ apportant une réelle amélioration sur ces deux dernières. Nous pouvons ainsi déduire que prendre des décisions partielles basées sur le plus court chemin n'est pas la stratégie la plus pertinente, même lorsque elle associée à une autre stratégie comme dans le cas de $PCPC$. Nous utiliserons donc $PCPA$ pour les expérimentations suivantes.

Groupe \ Politique	PCPA	PCPB	PCPC
A	5.11%	4.54%	3.84%
B	5.13%	5.06%	5.00%

Tableau 5.4 – Performance moyenne de $PCPA$, $PCPB$ et $PCPC$ sur les deux groupes d'instances

Comparaison HGA / HG

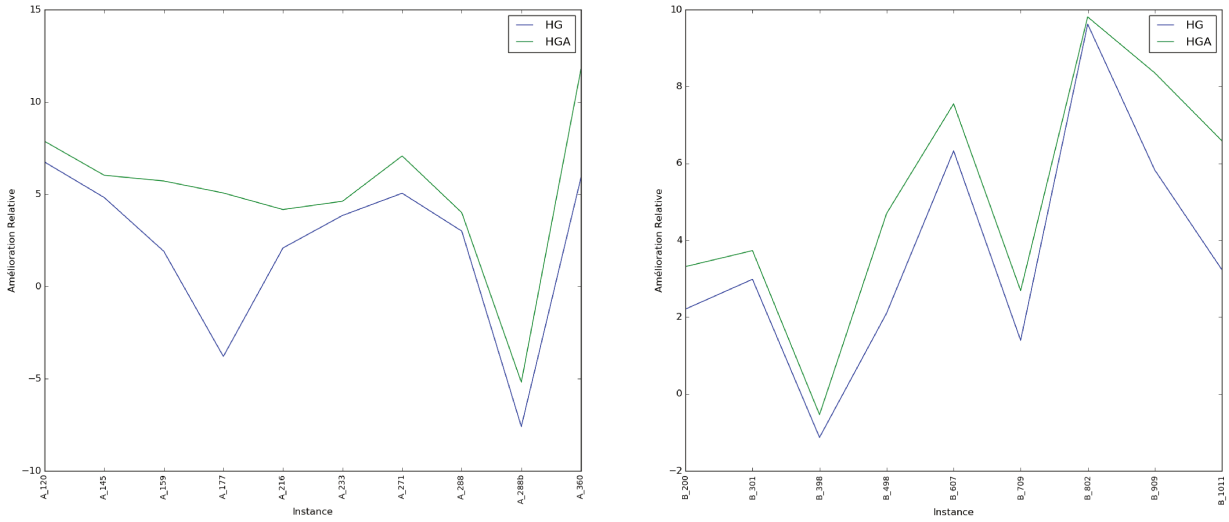
L'objectif ici est d'évaluer l'apport de l'algorithme HGA (algorithme 6) sur la version basique HG (algorithme 4). Nous avons effectué des simulations sur les deux groupes d'instances avec HG et HGA. Les deux heuristiques utilisent les valeurs de TG déterminées dans la section 5.7.2. Les paramètres spécifiques à l'algorithme HGA utilisés sont ceux déterminés aux sections précédentes (TG, $GTAB$ et politique de choix du meilleur chemin). La figure 5.7 présente les résultats pour les deux groupes d'instances.

Sur chaque groupe, HGA obtient de meilleurs résultats sur toutes les instances. L'amélioration moyenne relative de HGA sur HG est de 80% pour le groupe A et 57% pour le groupe B. HGA apporte donc une amélioration très significative sur HG. Ce constat s'explique facilement par le fait que la contrainte de non-retour imposée sur l'algorithme 5 limitant le champ de recherche de HG est atténuée par le caractère récursif de HGA.

Evaluation de l'impact de l'heuristique de réinsertion HGA

Après avoir trouvé une configuration efficace pour l'algorithme HGA, nous nous intéresserons à l'apport de l'heuristique en comparaison à l'algorithme online. Comme pour la section 5.7.1, nous nous intéresserons aux pourcentage de requêtes servies et à la durée totale du trajet des véhicules.

Le tableau 5.5 présente les résultats du groupe A avec 12, 13 et 14 véhicules pour chaque instance. O représente le pourcentage de requêtes servies avec l'algorithme online, $O+R$ est le pourcentage de requêtes servies lorsque l'algorithme HGA est utilisé, *Imp* est l'amélioration relative de l'algorithme HGA sur l'algorithme online en terme de requêtes servies et *ImpD* l'amélioration relative de l'algorithme HGA sur l'algorithme online en terme de durée des



(a) Groupe A

(b) Groupe B

Figure 5.7 – Comparaison de HGA et HG. L'axe des y représente l'amélioration relative par rapport à l'online et l'axe des x chaque instance du groupe associé.

Ve	12				13				14			
INS	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD
A_120	67.5	72.42	**7.28**	**-1.12**	74.17	80	**7.87**	0.58	75.83	81.08	**6.92**	**-0.77**
A_145	53.10	54.48	**2.60**	0.77	57.24	60.69	**6.02**	0.57	62.07	63.45	**2.22**	**-2.11**
A_159	60.38	62.26	**3.12**	1.79	66.04	69.81	**5.71**	4.18	65.41	71.07	**8.65**	8.23
A_177	42.37	48.02	**13.3**	7.18	44.63	46.89	**5.06**	**-0.26**	46.33	49.15	**6.1**	**-0.12**
A_216	43.52	42.13	-3.19	2.49	44.44	46.30	**4.17**	**-0.57**	45.37	47.69	**5.10**	1.53
A_233	51.07	54.51	**6.72**	3.58	55.8	58.37	**4.62**	4.27	60.95	60.6	-0.56	0.12
A_271	34.32	33.58	-2.15	0.93	36.53	39.11	**7.07**	**-1.72**	38.75	40.92	**5.62**	**-0.94**
A_288b	45.49	47.57	**4.50**	1.81	54.86	52.01	-5.19	0.46	55.55	56.81	**7.33**	2.96
A_288	34.03	33.02	-2.96	2.77	34.72	36.11	**4.0**	0.73	35.07	37.64	**2.25**	**-1.11**
A_360	26.39	26.39	0	0.93	28.33	31.67	**11.76**	**-1.46**	30.27	32.31	6.70	0.43
Avg.	45.82	47.43	**2.93**	2.11	49.68	52.1	**5.11**	0.68	51.56	54.07	**5.03**	0.48

Tableau 5.5 – Performance de l'algorithme HGA sur les instances du groupe A. Chaque instance est nommée G_N où G est le nom du groupe (A) et N le nombre de requêtes.

véhicules.

Comme nous l'avons vu précédemment, ces instances sont très difficiles. Cependant, l'algorithme HGA permet d'améliorer presque toutes les instances, avec une amélioration relative de 5.11% (respectivement 2.93% et 5.03%) avec 13 véhicules (avec 12 et 14 véhicules respectivement). Ces résultats sont obtenus avec une durée totale des véhicules similaire pour 13 et 14 véhicules, ce qui montre que l'algorithme HGA permet de servir plus de requêtes avec des coûts de fonctionnement similaires. Les résultats montrent aussi qu'avec 13 véhicules, l'algorithme HGA permet d'insérer en moyenne autant de requêtes qu'avec l'algorithme online avec 14 véhicules, faisant mieux sur 7 instances. L'algorithme HGA est ainsi capable de gagner l'apport obtenu par 1 véhicule sur presque toutes les instances avec des coûts de fonctionnement

similaires à l'algorithme online avec 13 véhicules, démontrant son efficacité sur des instances très difficiles. On observe également que l'amélioration relative de l'algorithme HGA ne diminue pas lors du passage à 14 véhicules. Tout comme dans le cas de l'algorithme HDR, l'algorithme HGA permet de mieux exploiter l'ajout d'un nouveau véhicule.
L'algorithme HGA implique dans des rares cas une détérioration en terme de pourcentage de requêtes servies. Sur les 5 situations où ce phénomène est constaté on observe systématiquement une augmentation de l'utilisation des véhicules, ce qui tend à confirmer que l'algorithme HGA insère au cours du service des requêtes qui rendent plus difficiles l'insertion des futurs clients.

Le tableau 5.6 présente les résultats du groupe B avec 5, 6 et 7 véhicules pour chaque instance. L'algorithme HGA apporte une amélioration sur chaque instance. Avec 6 véhicules, l'algorithme HGA donne une amélioration moyenne de 5.14% avec une durée totale des tournées supérieure de 2.43% en comparaison avec l'algorithme online, pour atteindre en moyenne 85.39% de requêtes servies. L'algorithme HGA avec 6 véhicules donne de meilleurs résultats que l'algorithme online seul avec 7 véhicules sur 7 instances avec en moyenne une augmentation des coûts de 2.43% en comparaison avec l'algorithme online. C'est un gain considérable pour le transporteur. L'amélioration relative de l'algorithme HGA est de 8.49% pour 5 véhicules, 5.14% pour 6 véhicules et 5.55% pour 7 véhicules tandis que les performances de l'algorithme online sont de 73.54%, 81.38% et 83.86% pour 5, 6 et 7 véhicules respectivement.

Ve	5				6				7			
INS	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD	O	O+R	Imp	ImpD
B_200	84.5	88.5	**4.73**	6.00	90.5	93.5	**3.31**	3.59	89.5	91.5	**2.23**	1.71
B_301	82.06	89.37	**8.91**	6.85	89.04	93.36	**3.73**	0.61	91.03	93.02	**2.19**	**-3.60**
B_398	86.93	88.69	**2.02**	0.88	88.94	88.47	-0.54	**-2.30**	87.44	92.46	**5.75**	7.96
B_498	76.51	84.60	**10.58**	6.47	85.94	89.98	**4.70**	4.10	87.35	92.70	**6.11**	**-0.36**
B_607	72.32	78.98	**9.20**	0.50	80.72	86.82	**7.55**	5.14	84.51	90.12	**6.63**	2.18
B_709	69.25	76.73	**10.79**	3.93	80.82	82.99	**2.69**	2.83	84.34	88.17	**4.53**	3.90
B_802	67.83	73.48	**8.33**	2.28	72.57	79.69	**9.81**	2.40	78.05	84.71	**8.53**	2.39
B_909	61.45	69.90	**13.75**	0.89	73.68	79.83	**8.36**	3.30	76.98	82.62	**7.32**	3.33
B_1011	61.03	65.96	**8.09**	0.65	70.23	74.86	**6.59**	2.15	75.57	80.62	**6.69**	1.35
Avg.	73.54	79.58	**8.49**	3.16	81.38	85.39	**5.14**	2.43	83.86	88.44	**5.55**	2.10

Tableau 5.6 – Performance de l'algorithme HGA sur les instances du groupe B. Chaque instance est nommée G_N où G est le nom du groupe (B) et N le nombre de requêtes.

5.8 Synthèse des expérimentations sur HG et HGA

Les expérimentations ont clairement montré que TG a un impact très élevé sur l'algorithme HGA, ce qui montre la nécessité de choisir une valeur appropriée. La politique de tabou n'a en revanche que très peu d'impact. Les expérimentations ont aussi mis en évidence que la partie récursive de HGA fournit un apport très conséquent sur l'algorithme HG, la version non récursive. HGA et TG sont donc deux améliorations fondamentales qui permettent d'obtenir de bons résultats sur les deux groupes d'instances. L'algorithme HGA permet ainsi d'obtenir avec le nombre par défaut de véhicules des performances meilleures que l'algorithme online avec

un véhicule supplémentaire sur la grande majorité des instances, même sur les instances très difficiles du groupe A.

5.9 Comparaison entre HDR et HGA

L'objectif de cette section est de comparer les deux algorithmes présentés dans ce chapitre : HDR et HGA. Le tableau 5.7 compare les résultats obtenus aux sections 5.7.1 et 5.7.2 pour le groupe A. *Imp* représente l'amélioration relative par rapport à l'algorithme online et *ImpD* l'amélioration relative par rapport à l'algorithme online en terme de durée des tournées. *VE* représente le nombre de véhicules et *INS* les différentes instances.

Concernant le pourcentage de requêtes servies, l'algorithme HDR obtient de meilleurs résultats sur 2/3 des instances et permet une légère amélioration des performances moyennes. Sur 13 véhicules, l'algorithme HGA obtient les meilleures performances sur la moitié des instances avec des performances moyennes supérieures (plus de 10% d'amélioration par rapport à l'algorithme HDR). L'algorithme HGA permet d'économiser 1 véhicule sur 7 instances tandis que l'algorithme HDR permet d'économiser 1 véhicule sur 5 instances (cf sections 5.7.1 et 5.7.2). L'algorithme HDR est en revanche meilleur en moyenne sur 12 et 14 véhicules ainsi que sur la majorité des instances. Concernant l'utilisation des véhicules, on observe que l'algorithme HDR obtient des performances moyennes qui sont toujours meilleures, le nombre d'instances où l'algorithme HGA obtient une durée inférieure étant beaucoup plus faible. En résumé, l'algorithme HDR permet de servir un nombre proche ou supérieur de requêtes par rapport à l'algorithme HGA tout en assurant des coûts de fonctionnement significativement inférieurs.

	Imp						ImpD					
VE	12		13		14		12		13		14	
INS	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA
A_120	**10.12**	7.28	**10.56**	7.87	**10.11**	6.92	-0.40	**-1.12**	**-1.40**	0.58	0.74	**-0.77**
A_145	**4.81**	2.60	**6.14**	6.02	-0.89	**2.22**	1.06	**0.77**	**-0.73**	0.57	**-2.91**	-2.11
A_159	**3.96**	3.12	3.71	**5.71**	**10.00**	8.65	3.21	**1.79**	**2.57**	4.18	**2.04**	8.23
A_177	10.13	**13.3**	4.56	**5.06**	1.95	**6.1**	**4.88**	7.18	**-0.86**	-0.26	**-2.55**	-0.12
A_216	**-1.70**	-3.19	1.46	**4.17**	**6.43**	5.10	**1.04**	2.49	-0.43	**-0.57**	**0.38**	1.53
A_233	2.94	**6.72**	**5.15**	4.62	**0.85**	-0.56	**0.62**	3.58	**3.57**	4.27	**-0.38**	0.12
A_271	**1.93**	-2.15	**7.68**	7.07	**8.19**	5.62	**-0.21**	0.93	-3.90	**-1.72**	-0.91	**-0.94**
A_288	**0.10**	-2.96	2.20	**4.0**	**8.12**	2.25	**0.88**	2.77	**0.39**	0.73	0.95	**-1.11**
A_288b	**12.52**	4.50	**-1.65**	-5.19	3.56	**7.33**	**-0.58**	1.81	**-0.14**	0.46	**-0.89**	2.96
A_360	**4.11**	0	5.88	**11.76**	**7.52**	6.70	**0.30**	0.93	**-2.86**	-1.46	0.48	**0.43**
Avg.	**4.89**	2.93	4.57	**5.11**	**5.58**	5.03	**1.08**	2.11	**-0.38**	0.68	**-0.31**	0.48

Tableau 5.7 – Comparaison de HDR et HGA pour le groupe A.

Le tableau 5.8 compare les résultats obtenus aux sections 5.7.1 et 5.7.2 pour le groupe B. Les résultats sont beaucoup plus explicites que pour le groupe A : l'algorithme HDR obtient de meilleures performances moyennes en terme de requêtes servies pour 5, 6 et 7 véhicules. L'algorithme HDR fait en effet mieux sur toutes les instances pour 5 véhicules, sur 7 instances pour 6 véhicules et sur 8 instances pour 7 véhicules. Ces résultats sont obtenus avec une augmentation de la durée significativement plus faible que pour l'algorithme HGA pour la grande majorité des instances. En résumé, l'algorithme HDR permet de servir plus de requêtes avec

des coûts de fonctionnement moindres et permet d'économiser 1 véhicule sur plus d'instances que l'algorithme HGA (cf sections 5.7.1 et 5.7.2).

	Imp						ImpD					
VE	5		6		7		5		6		7	
INS	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA	HDR	HGA
B_200	**5.27**	4.73	2.49	**3.31**	**2.79**	2.23	**5.27**	6.00	**-2.42**	3.59	1.93	**1.71**
B_301	**10.97**	8.91	3.66	**3.73**	**3.43**	2.19	**3.95**	6.85	**-1.61**	0.61	-0.80	**-3.60**
B_398	**4.34**	2.02	**2.54**	-0.54	5.08	**5.75**	1.29	**0.88**	0.39	**-2.30**	**1.99**	7.96
B_498	**12.34**	10.58	**5.70**	4.70	**7.47**	6.11	**2.42**	6.47	**2.40**	4.10	0.97	**-0.36**
B_607	**13.64**	9.20	**9.47**	7.55	**8.34**	6.63	0.85	**0.50**	**1.88**	5.14	**-0.38**	2.18
B_709	**12.95**	10.79	**4.92**	2.69	**5.53**	4.53	**0.87**	3.93	**0.96**	2.83	**1.32**	3.90
B_802	**10.58**	8.33	**14.05**	9.81	**11.02**	8.53	**-0.29**	2.28	**2.08**	2.40	**0.06**	2.39
B_909	**17.29**	13.75	**10.45**	8.36	**11.00**	7.32	**0.48**	0.89	**1.17**	3.30	**2.90**	3.33
B_1011	**13.06**	8.09	**10.37**	6.59	**8.65**	6.69	**-0.02**	0.65	**0.44**	2.15	**0.38**	1.35
Avg.	**11.16**	8.49	**7.07**	5.14	**7.04**	5.55	**1.65**	3.16	**0.59**	2.43	**0.93**	2.10

Tableau 5.8 – Comparaison de HDR et HGA pour le groupe B.

5.10 Conclusion

Dans ce chapitre, nous avons étudié deux heuristiques de réinsertion : HDR et HGA. Ces heuristiques contiennent très peu de paramètres libres à déterminer. Nous avons procédé à une analyse de l'impact de ces paramètres et avons pu déterminer les meilleurs valeurs sur nos instances. Les deux heuristiques ont montré leur capacité à augmenter le nombre de requêtes servies par le système tout en maintenant des coûts de fonctionnement proches de ceux de l'online. De plus, le temps de calcul de 3 secondes choisi permet à ces heuristiques d'être utilisées sans nuire à la qualité de service des clients. Les deux algorithmes proposés permettent d'améliorer les performances du système tout en maintenant une qualité de service élevée, ce qui les rend intégrables en production sur des services réels. Nous avons également procédé à une comparaison entre les deux heuristiques et montré que l'algorithme HDR permet en moyenne de servir plus de requêtes avec des coûts de fonctionnement inférieurs. C'est donc l'algorithme HDR qui sera choisi en priorité pour être utilisé sur des services réels.

Les heuristiques d'insertion présentées ont ainsi démontré qu'il existe une marge d'optimisation intéressante à gagner par rapport à l'algorithme online. Ces dernières n'apportent cependant une optimisation supplémentaire que lorsqu'une requête est réinsérée et ne profitent pas du temps disponible entre l'apparition des différentes requêtes. Dans le chapitre suivant, nous présentons une heuristique permettant de tirer profit de ce temps libre et d'optimiser les tournées en continu lors du service.

Chapitre 6

Module Offline

6.1 Introduction

Dans les chapitres 4 et 5, nous avons présenté des algorithmes permettant une réponse temps réel aux utilisateurs. De par leur conception et leur but, ces algorithmes ne profitent pas du temps disponible durant un service ou entre les réservations des clients pour optimiser les tournées. Pour tenter de pallier ce manque, nous présenterons dans ce chapitre le module offline, dont l'objectif est de réarranger les tournées pendant le service pour servir le plus de requêtes possible. Après avoir donné les détails de la problématique en section 6.2, nous présenterons en section 6.3 l'algorithme principal du module, basé sur une métaheuristique : l'ALNS. La section 6.4 exposera un moyen de régler les hyper-paramètres de l'heuristique. Enfin, des expérimentations sur les instances présentées en section 3.3.2 seront menées dans les sections 6.5.1 et 6.5.2 pour évaluer le comportement du module lorsqu'il est utilisé sur les requêtes en avance ou en continu durant le service. La section 6.6 présentera la conclusion du chapitre.

6.2 Présentation de la problématique

Dans les chapitres 4 et 5, nous avons explicité le fonctionnement des modules online et de réinsertion qui permettent de répondre en temps réel aux requêtes des utilisateurs. Cependant, aucun de ces modules n'exploite le temps disponible entre deux évènements du système, comme l'arrivée d'un véhicule à un noeud ou l'arrivée d'une nouvelle requête. L'objectif du module offline est d'optimiser les tournées en utilisant au maximum ce temps disponible. Pour ce faire, il cherche des solutions dans le voisinage des tournées en déplaçant les requêtes des utilisateurs déjà insérés dans la solution. Le module offline peut-être utilisé de deux manières différentes :

— Sur les requêtes en avance : le module est lancé un peu avant le début du service sur les requêtes faites en avance.
— En continu : le module est lancé entre chaque évènement (arrivée/départ d'un véhicule à un noeud, nouvelle requête) d'un service.

6.2.1 Données d'entrée et interaction avec le module de dispatch

C'est le module de dispatch (cf figure 2.3) qui est en charge de lancer le module offline. Ce dernier reçoit les tournées actuelles (i.e la solution actuelle) et le temps de recherche maximal. Ce temps est utilisé en simulation où l'heure du prochain évènement est toujours connu, ce

qui permet de calculer le temps libre entre l'instant actuel et le prochain évènement. Dans une application temps réel, ce temps est inconnu et le module doit recevoir des signaux asynchrones du module de dispatch pour être informé des évènements. Dans ce chapitre, nous supposerons toujours que le temps de recherche maximal est connu. Pour plus de détails, voir le chapitre 7. Une fois sa recherche terminée, le module offline transmet au module de dispatch la meilleure solution trouvée durant la recherche. Cette solution devient la solution courante pour la suite du service.

6.2.2 Contraintes et défis

Le problème auquel le module offline fait face à chaque lancement peut se résumer de la manière suivante : étant donné une solution initiale et un temps maximal, trouver une solution avec un coût (déterminé par la fonction objectif) minimal. Le module offline dispose d'un nombre fini de tournées dans lesquelles il a la possibilité de déplacer les utilisateurs déjà insérés. Il doit cependant respecter leurs contraintes horaires (fenêtre horaire et gamma) et les contraintes de capacité des véhicules. L'objectif final du module offline est de permettre de servir plus de requêtes sur l'ensemble du service. Toutefois, seule une partie des requêtes utilisateurs est connue à chaque lancement du module offline, ce qui rend l'optimisation directe du nombre de requêtes acceptées durant le service impossible. Le module offline doit donc être capable de créer des tournées permettant d'insérer le plus de requêtes possible en travaillant avec des informations partielles.

6.2.3 Objectifs opérationnels et scientifiques

Le développement d'un module offline efficace est très important pour permettre de servir le plus de requêtes possible dans le cadre des services opérés par Padam. Il permet de plus d'obtenir des tournées de haute qualité et d'économiser des coûts sur les déplacements des véhicules. Comme nous l'avons montré dans l'état de l'art (section 3.2.4), peu de travaux ont considéré l'utilisation d'une procédure de post-optimisation (i.e module offline) couplée avec une heuristique d'insertion. Les travaux rencontrés ne considèrent qu'un seul véhicule ([Coslovich et al., 2006]), une flotte infinie ([Häll et al., 2012]) ou des contraintes horaires souples ([Beaudry et al., 2010]). Aucun des travaux rencontrés ne proposent une analyse de l'impact d'un module offline en avance et/ou en continue sur le pourcentage de requêtes servies. De plus, les approches basées sur les heuristiques à grand voisinage n'ont pas été utilisées dans ce cadre pour le DARP dynamique. Le problème sous-jacent à la conception du module offline présente donc un intérêt à la fois sur les plans opérationnel et scientifique.

6.3 Principe général de l'algorithme offline basé sur la méthode ALNS

L'algorithme principal du module offline est basé sur la métaheuristique Adaptative Large Neighborhood Search (ALNS), qui a été utilisée avec succès dans des problèmes de tournées de véhicules ([Ropke and Pisinger, 2006], [Li et al., 2016]) et sur des instances statiques du DARP ([Li et al., 2016]).
L'ALNS est une amélioration de la métaheuristique Large Neighborhood Search (LNS) introduite dans [Shaw, 1998]. L'idée derrière LNS est de détruire une grande partie de la solution pour la reconstruire ensuite. Un opérateur de destruction est utilisé pour choisir puis enlever

un ensemble de requêtes et un opérateur de reconstruction pour les réinsérer intelligemment dans la solution. En partant d'une solution initiale, l'heuristique applique à chaque itération l'opérateur de destruction et de reconstruction jusqu'à atteindre un critère d'arrêt. Le voisinage défini par l'opérateur de destruction étant en général énorme, il ne peut être parcouru dans son intégralité comme dans le cas d'une recherche locale. L'heuristique va donc de voisin en voisin par 'pas de géant' avec l'espoir de balayer toutes les zones intéressantes de l'espace des solutions. Une banque de requêtes est mise à jour avec toutes les requêtes non insérées à chaque itération. Ceci permet à l'heuristique d'explorer des solutions non faisables tout au long de la recherche.

L'ALNS, à l'origine introduite par [Ropke and Pisinger, 2006], reprend et améliore le concept de LNS. La principale amélioration consiste à utiliser plusieurs opérateurs de destruction/reconstruction. L'algorithme 8 montre le pseudo-code associé à l'ALNS que nous avons implémentée. A chaque itération, une étape de destruction (lignes 10 et 11) et de perturbation (L12) sont appliquées sur la solution actuelle. Une décision est ensuite prise quand à l'acceptation de la nouvelle solution (ligne 13) qui peut devenir la nouvelle meilleure solution connue (L16). Au fur et à mesure de la recherche, une fonction de rétribution est utilisée pour favoriser ou pénaliser les différents opérateurs (lignes 7, 14 et 17). Cette fonction est basée sur les performances des opérateurs aux itérations précédentes. Nous allons désormais présenter plus en détail les différentes composantes de notre ALNS.

Algorithm 8 ALNS

Entrée : solution actuelle s, temps maximum MT
Sortie : meilleure solution trouvée s_{best} si elle existe

1: niter$\leftarrow$ 0
2: $s_{best} \leftarrow s$
3: $s_{current} \leftarrow s$
4: T$\leftarrow$ $TrouveTemperatureInitiale()$
5: **while** niter $<$ 25000 et Temps écoulé $< MT$ **do**
6: niter_seg $\leftarrow$ 0
7: score_seg $\leftarrow$ $DictionnnaireVide()$
8: **while** niter_seg $<$ seg et Temps écoulé $< MT$ **do**
9: op_sel, op_per$\leftarrow$ $SelectOperateurs()$
10: k$\leftarrow$ $SelectNombreRequetes()$
11: L$\leftarrow$ $SelectRequetes$(s, op_sel, k)
12: $s_{new} \leftarrow PerturbeSolution$(s, op_per, L)
13: $s_{current} \leftarrow DecideAcceptation(s_{new}$, T)
14: score$\leftarrow$ $CalculScore(s_{new})$
15: **if** $f(s_{current}) < f(s_{best})$ **then**
16: $s_{best} \leftarrow s_{current}$
17: score_seg$\leftarrow$ $MetaJourScoreSegment$(ops, score)
18: $MiseAJourTemperature(T)$
19: niter_seg $\leftarrow$ niter_seg +1
20: **end while**
21: niter$\leftarrow$ niter + niter_seg
22: $MetAJourDistributionOperateurs$(score_seg)
23: **end while**

6.3.1 Fonction objectif

La métrique utilisée pour mesurer la qualité de la solution est la durée totale des tournées (cf section 3.2.1). Il est néanmoins nécessaire de prendre également en compte la taille de la banque de requêtes lors de la recherche, que nous appellerons T_b. La fonction objectif s'écrit donc :

$$f(s) = f_d(s) + w_{buff} * T_b$$

où w_{buff} est un paramètre libre déterminant le coût engendré par le fait d'avoir une requête non insérée dans la solution. La valeur du paramètre peut-être arbitrairement large, auquel cas la recherche n'accepte jamais de solution non faisable.

6.3.2 Conditions initiales et conditions d'acceptation

La solution initiale correspond à la solution à l'instant où est lancé l'offline, c'est à dire l'état des tournées créées par l'online avec les précédentes requêtes. (Dans le cas des problèmes statiques, la solution initiale est souvent construite à partir d'une heuristique simple ([Ropke and Pisinger, 2006])).
Un critère de recuit simulé classique est utilisé pour décider de l'acceptation d'une nouvelle solution (ligne 13 de l'algorithme 8). Cela signifie qu'étant donné la solution actuelle s et f la fonction objectif, une nouvelle solution $s^{'}$ est acceptée avec probabilité $\exp \frac{f(s)-f\left(s^{'}\right)}{T}$ où T est la température et f la fonction objective. Une meilleure solution est donc nécessairement acceptée et une solution dégradant la solution actuelle peut aussi être acceptée, donnant ainsi la possibilité d'échapper à des minima locaux. La température part d'une valeur initiale T_0 et décroit à chaque itération en utilisant la formule $T_{n+1} = c * T_n$, avec $c < 1$ qui représente le taux de refroidissement. La valeur de T_0 est dépendante de l'instance et conditionne l'efficacité de la recherche. Elle est déterminée de telle sorte qu'une solution 10% pire que la solution actuelle soit acceptée avec probabilité 0.5 :

$$T_0 = \frac{f(s_0) \times 0.1}{log(2)}$$

avec s_0 la solution initiale (ligne 4 de l'algorithme 8).

6.3.3 Opérateurs de destruction

A chaque itération, un nombre k de requêtes est enlevé (ligne 11 de l'algorithme 8). k est choisi aléatoirement entre $k_{min} = min_r * T_{request}$ et $k_{max} = max_r * T_{request}$ où $T_{request}$ est le nombre total de requêtes non servies et $max_r, min_r \in]0; 1]$ deux paramètres libres (ligne 10 de l'algorithme 8). Une requête est appelée non-servie lorsque son pickup n'a pas encore été servi par le véhicule associé. Si la banque de requêtes contient k_b requêtes, les $\min(k_b, k)$ premières requêtes de la banque sont automatiquement choisies. k est ensuite mis à jour : $k = \max(k - k_b, 0)$. Nous utilisons 5 opérateurs de destruction, les 3 premiers étant inspirés de [Ropke and Pisinger, 2006] :

Random : Choisit k requêtes aléatoirement.

Worst : Choisit les k requêtes ayant les meilleurs "saving", le "saving" d'une requête étant la différence entre la valeur objectif de la solution actuelle et celle de la solution une fois

la requête enlevée. Pour augmenter la diversification, l'opérateur est randomisé : toutes les requêtes de LI sont triées en ordre décroissant de "saving" dans une liste L. Un nombre aléatoire y entre 0 et 1 est tiré et la requête en position $\lfloor y^{p_w}|L| \rfloor$ (où $|L|$ est la taille de la liste L et p_w un paramètre) est choisie. Cette procédure est répétée jusqu'à ce que k requêtes aient été choisies.

Relatedness : Choisi au hasard une requête et sélectionne $k-1$ requêtes similaires. La mesure de similarité entre les requêtes i et j est définie comme suit :

$$r_t \left(t_{p_i,p_j} + t_{d_i,d_j}\right) + r_d \left(|u_{p_i} - u_{p_j}| + |u_{d_i} - u_{d_j}|\right)$$

où p_i et d_i sont respectivement les noeuds de pickup et dropoff, et u_{p_i} et u_{d_i} les heures de service aux pickup et dropoff de le requête i. Plus la mesure de similarité a une valeur faible, plus les deux requêtes sont considérées comme similaires. Toutes les requêtes de LI sont triées en ordre croissant de similarité dans une liste L. Un nombre aléatoire entre 0 et 1 est tiré et la requête en position $\lfloor y^{p_r}|L| \rfloor$ (où $|L|$ est la taille de la liste L) est choisie. Cet opérateur comprend 3 paramètres libres : $r_t \in [0;1], r_d \in [0;1]$ tels que $r_t + r_d = 1$ et p_r.

Strong Overlap Relatedness : Nous avons proposé une adaptation de l'opérateur précédent où la mesure de similarité est basée sur les fenêtres horaires des requêtes. Ceci est motivé par le fait que la contrainte temporelle est de loin la plus forte dans notre problème. L'opérateur fonctionne de la manière suivante : une première réservation est choisie aléatoirement et toutes les requêtes considérées comme similaires vont être enlevées à leur tour. La mesure de similarité est différente de l'opérateur précédent et basée sur la notion de ratio de chevauchement. Si $T_{p_i} = \left[\underline{p}_i, \overline{p}_i\right]$ est la fenêtre horaire du pickup d'une requête i et $T_{d_i} = \left[\underline{d}_i, \overline{d}_i\right]$ la fenêtre horaire de son dropoff, le ratio est calculé de la manière suivante :

$$r_{ij} = \begin{cases} 0 & \text{si } T_{p_i} \cap T_{p_j} = \emptyset \text{ OU } T_{d_i} \cap T_{d_j} = \emptyset \\ \frac{inter(T_{p_i},T_{p_j}) + inter(T_{d_i},T_{d_j})}{\left(\overline{p}_i - \underline{p}_i\right) + \left(\overline{d}_i - \underline{d}_i\right)} & \text{sinon} \end{cases}$$

où $inter(a,b)$ est la taille de l'intersection des intervalles a et b. Etant donné 2 requêtes i et j, i est similaire à j si et seulement si leur ratio de chevauchement excède un seuil prédéfini noté r_s. Cet opérateur sélectionne des requêtes qui se ressemblent et qui pourront donc par la suite être échangées pour créer des tournées plus cohérentes. Cet opérateur ne tient pas compte de k et sélectionne toutes les requêtes ressemblantes à la première requête, choisie aléatoirement.

Random Ride : Pour notre problème, nous avons également proposé un autre opérateur de destruction qui consiste à sélectionner toutes les requêtes de k_r véhicules choisis aléatoirement. Cet opérateur ne tient pas compte de k et sélectionne toutes les requêtes associées aux véhicules sélectionnés.

6.3.4 Opérateurs de perturbation

Après avoir sélectionné les requêtes à enlever, les opérateurs de perturbation sont utilisés pour les enlever et les réinsérer (ligne 12 de l'algorithme 8). Un opérateur peut enlever toutes les requêtes en une fois et les réinsérer une par une ou les enlever une par une puis les réinsérer immédiatement. Appelons L la liste des requêtes à enlever. Chaque insertion de chaque requête est associée à un coût, qui est la valeur de la fonction objectif si l'insertion est réalisée.

Deep Greedy : Enlève toutes les requêtes de la liste L en une fois. Calcule les insertions de toutes les requêtes dans toutes les tournées. Applique la meilleure insertion possible. Recalcule les nouvelles insertions et réitère la procédure jusqu'à ce que toutes les requêtes aient été insérées ou que plus aucune insertion ne soit faisable.

Regret : Enlève toutes les requêtes de la liste L en une fois. Calcule les insertions de toutes les requêtes dans toutes les tournées. Insère la requête de L qui a le plus grand regret. Le regret r_i d'une requête i est défini de la manière suivante : soit c_{ik} le coût associé à la k-ème meilleure insertion. Alors $r_i = c_{i2} - c_{i1}$. Le regret évalue en quelque sorte le prix qui pourrait être payé si la requête n'était pas insérée maintenant. Si une seule insertion est faisable pour i, le regret est égal à une valeur arbitrairement grande. La requête i sélectionnée est celle qui a le plus grand regret et sa meilleure insertion est réalisée. L'objectif du regret est ainsi de réaliser en priorité les insertions difficiles. Une fois la requête insérée, toutes les insertions sont recalculées et la procédure réitérée jusqu'à ce que toutes les requêtes aient été insérées ou que plus aucune insertion ne soit faisable.

Parallel Regret : Similaire à l'opérateur de regret mais avec une mesure de regret différente, comme présentée dans [Diana and Dessouky, 2004]. L'idée est de trouver pour chaque requête $i \in L$ sa meilleure insertion dans chaque véhicule k. Le coût de l'insertion est noté c_{ik}. Une matrice est ainsi construite où chaque ligne représente une requête et chaque colonne un véhicule. Si une requête n'a pas d'insertion possible dans une route, la valeur associée dans la matrice est choisie arbitrairement grande. Le regret est ensuite donné par

$$\sum_k \left(c_{ik} - \min_j c_{ij} \right)$$

La requête avec le plus grand regret est ensuite choisie et insérée dans sa meilleure position.

Basic Greedy : Enlève les requêtes une par une de la liste L. Pour chaque requête enlevée, calcule toutes les insertions faisables de la requête dans les tournées et applique la meilleure.

Les requêtes qui n'ont pas pu être insérées par l'opérateur choisi sont ajoutées à la banque de requêtes.

6.3.5 Mise à jour adaptative

Un mécanisme de roulette ("roulette wheel selection") est utilisé pour sélectionner les opérateurs (ligne 9 de l'algorithme 8). Les opérateurs de destruction et réparation sont choisis indépendemment à chaque itération selon une distribution de probabilité mise à jour au fur et à mesure de la recherche (L22). Au début de la recherche, la distribution de probabilité des opérateurs est uniforme. La recherche est ensuite divisée en segments de taille *seg* itérations (L8). A la fin de chaque itération, le score pour chaque opérateur utilisé est augmenté selon 4 paramètres (lignes 14 et 17) :

σ_1 si une nouvelle meilleure solution faisable est obtenue

σ_2 si une nouvelle meilleure solution non faisable (i.e avec requêtes dans la banque) est obtenue

σ_3 si une solution meilleure que l'actuelle est obtenue

σ_4 si une nouvelle solution moins bonne que l'actuelle est acceptée

A la fin de chaque segment, la probabilité de chaque opérateur est mise à jour en fonction des performances des opérateurs sur le segment comme suit (ligne 22) :

$$p_i^{s+1} = p_i^s (1 - r) + r \frac{\pi_i}{n_i}$$

où π_i est le score cumulé de l'opérateur i durant le segment s et n_i le nombre de fois que l'opérateur a été utilisé. r est un paramètre appelé facteur de réactivité.

6.4 Réglage des paramètres

L'objectif de cette section est de trouver des bonnes valeurs pour les paramètres libres de notre ALNS. Les paramètres à régler sont les suivants :

— max_r, min_r : proportion maximale et minimale de requêtes à enlever à chaque itération
— c : taux de refroidissement du recuit simulé
— p_r, p_w : paramètres de randomisation des opérateurs "Worst" et "Relatedness"
— r_t, r_d : poids associés au calcul de la similarité dans l'opérateur de Relatedness
— r_s : ratio de similarité dans l'opérateur de "Strong Overlap Relatedness"
— k_r : nombre maximal de tournées à détruire pour l'opérateur "Random Ride"
— seg : taille des segments de l'ALNS
— r : facteur de réaction dans la mise à jour des poids
— $buff$: poids associé à chaque requête de la banque de requêtes dans le calcul de l'objectif
— σ_1, σ_2, σ_3, σ_4 : scores possibles de la nouvelle solution

Pour déterminer un bon ensemble de paramètres, nous utilisons Irace ([López-Ibáñez et al., 2016b]), un outil de configuration automatique destiné à trouver des bonnes configurations pour les paramètres libres des algorithmes d'optimisation. La section suivante présente une vue d'ensemble d'Irace.

6.4.1 Irace

L'objectif d'un algorithme de configuration automatique est de trouver, durant la phase d'entraînement, une configuration des paramètres qui minimise une mesure de coût sur l'ensemble des instances rencontrées durant la phase de production. L'objectif final est que la configuration trouvée se généralise à des instances similaires mais non rencontrées durant l'entraînement. La mesure de coût assigne une valeur à un test d'une configuration de paramètres (représentée par un vecteur) sur une instance. Dans notre exemple, ce coût correspond à la durée totale des tournées de la meilleure solution trouvée par l'ALNS. De manière plus formelle, l'objectif est de trouver

$$\min_{\theta \in \Theta} \mu_\theta = \int c dP_C (c, |\theta, i) \, dP_I (i) \tag{6.1}$$

où Θ représente l'ensemble de tous les paramètres possibles de l'ALNS, θ un élément de Θ et i une instance. P_I est la mesure de probabilité générant les instances. Dans notre situation, elle est essentiellement influencée (pour un territoire donné) par la loi de probabilité d'apparition des requêtes et de l'offre de service (disposition des véhicules etc...). P_C représente la probabilité

d'obtenir un coût donné étant donné un vecteur de paramètre et une instance. Elle modélise le fait que plusieurs exécutions de l'algorithme sur une instance peuvent donner des résultats différents, ce qui est le cas de l'ALNS (et de beaucoup de méta-heuristiques). P_I et P_C sont bien sûr inconnues. Cependant, tout ce dont nous avons besoin (comme dans les méthodes de Monte-Carlo) c'est d'être en mesure d'échantillonner des réalisations de ces lois. Déterminer θ_{opt} qui minimise 6.1 ne signifie pas forcément que cet ensemble de paramètres aura la meilleur performance sur chaque instance mais que les performances moyennes sont les meilleures. L'objectif de Irace est d'essayer d'évaluer θ_{opt}.

Etant donné un Budget d'entraînement B (temps total d'entraînement ou nombre maximum d'exécutions de l'algorithme), une approche naïve serait de diviser Θ en un ensemble fini de configurations et de consacrer une part égale du budget à chaque configuration. La meilleure serait la configuration qui obtient la meilleure moyenne sur les lancers effectués. L'inconvénient de cette méthode réside dans le fait qu'une grande partie du budget d'entraînement risque d'être utilisé sur des mauvaises configurations. Une approche plus intéressante serait de déterminer rapidement les mauvaises configurations et les éjecter de la recherche dès que possible. C'est l'idée principale de la notion de "race" : lorsque plusieurs vecteurs de paramètres sont en concurrence, des tests statistiques sont réalisés et les candidats apparaissant comme moins bons sont éjectés. La suite de la recherche se concentre ainsi uniquement sur les meilleures configurations.

La notion de "race" est au coeur du fonctionnement de Irace. Irace reçoit en entrée l'espace des paramètres possibles pour l'algorithme à optimiser ainsi qu'un ensemble d'instances sur lesquelles l'algorithme sera testé. Les principales étapes de la recherche sont les suivantes :

1. Échantillonner un ensemble de candidats suivant une certaine distribution (au départ uniforme)
2. Déterminer les meilleurs candidats ("race")
3. Mettre à jour la distribution d'échantillonage en fonction des meilleurs candidats

Ces 3 étapes sont répétées jusqu'à atteindre un critère d'arrêt, comme l'épuisement du budget initial par exemple. Pour plus de détails sur le fonctionnement de Irace, voir [López-Ibáñez et al., 2016b]. Un guide pour l'utilisation du package R associé est fourni dans [López-Ibáñez et al., 2016a].

6.4.2 Recherche des paramètres

L'objectif du réglage des paramètres est de trouver un ensemble de paramètres efficaces sur les instances rencontrées par le module offline. Les deux groupes correspondent à deux services indépendants avec chacun sa loi de génération (cf 6.1) et des meilleurs configurations de paramètres pouvant être différentes. L'entraînement sera donc fait de manière indépendante pour les deux groupes d'instances. 100 instances d'entraînement sont générées pour chaque groupe, de la manière suivante :

1. Un nombre de requêtes r est choisi aléatoirement entre 20 et la taille de l'instance ayant le plus grand nombre de requêtes.
2. r requêtes sont ensuite choisies aléatoirement parmi l'ensemble de toutes les requêtes de toutes les instances

3. Des tournées sont ensuite créées en envoyant les r requêtes choisies une par une au module online. Ces tournées sont les instances sur lesquels Irace va travailler, en servant de solution initiale aux exécutions de l'ALNS.

Une fois les instances générées, l'espace des paramètres doit être défini. Le tableau 6.1 présente les intervalles de variation que nous avons définis pour les différents paramètres libres de l'ALNS (similaires pour les deux groupes d'instances).

Paramètre	max_r	min_r	c	p_w
Valeur	$[0.05, 0.4]$	$[0.01, 0.05]$	$[0.995, 0.99975]$	$[\![1, 10]\!]$
Paramètre	r_t	r_d	p_r	r_s
Valeur	$[0.1, 0.9]$	$[0.1, 0.9]$	$[\![1, 10]\!]$	$[0.3, 0.9]$
Paramètre	k_r	seg	r	$buff$
Valeur	$[\![1, 3]\!]$	$[\![10, 100]\!]$	$[0.1, 0.7]$	$[\![500, 100000]\!]$
Paramètre	σ_1	σ_2	σ_3	σ_4
Valeur	$[\![1, 30]\!]$	$[\![1, 30]\!]$	$[\![1, 30]\!]$	$[\![1, 30]\!]$

Tableau 6.1 – Intervalle de variation des paramètres de l'ALNS

Avec cet espace des paramètres nous avons défini le budget maximal à 7h, qui représente le temps total passé dans les différentes exécutions de l'ALNS. Nous avons utilisé le mode parallèle d'Irace en lançant sur 7 threads. Le budget représentant le temps des exécutions de l'ALNS, ceci revient à un temps total d'entraînement d'environ 1 heure. Le tableau 6.1 présente les valeurs obtenues.

Paramètre	max_r	min_r	c	p_w
Valeur	0.18	0.03	0.9955	4
Paramètre	r_t	r_d	p_r	r_s
Valeur	0.30	0.70	6	0.38
Paramètre	k_r	seg	r	$buff$
Valeur	2	20	0.67	2400
Paramètre	σ_1	σ_2	σ_3	σ_4
Valeur	30	11	21	16

(a) Groupe A

Paramètre	max_r	min_r	c	p_w
Valeur	0.33	0.015	0.995	10
Paramètre	r_t	r_d	p_r	r_s
Valeur	0.53	0.46	3	0.54
Paramètre	k_r	seg	r	$buff$
Valeur	1	96	0.23	13000
Paramètre	σ_1	σ_2	σ_3	σ_4
Valeur	27	14	19	13

(b) Groupe B

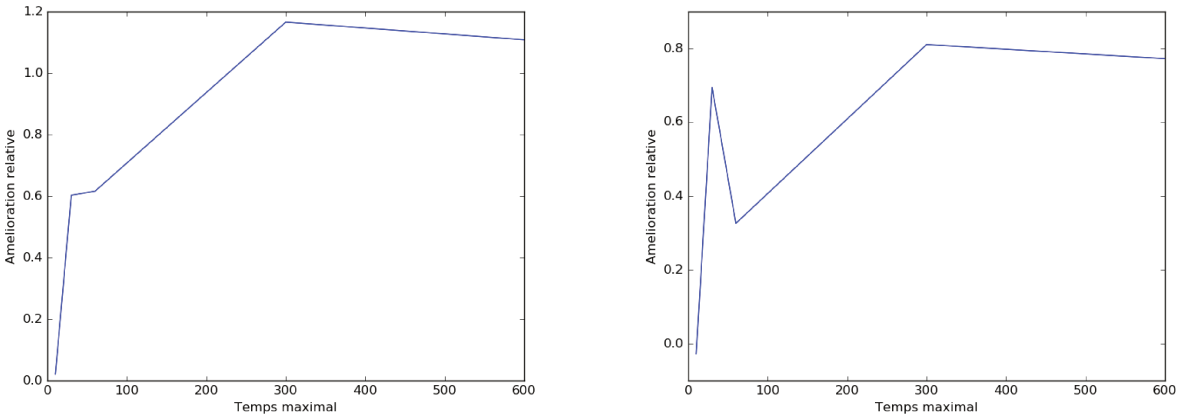
Figure 6.1 – Valeurs des paramètres de l'ALNS

6.5 Expérimentations

Toutes les simulations sont menées sur une machine équipée 32GB de RAM et un processeur quadri-core Intel Xeon E3-1245 cadencé à 3.40GHZ. Le module est implémenté entièrement en Python/Cython [1]. Le module offline étant basé sur une heuristique stochastique, les expérimentations sont lancées 10 fois sur chaque instance pour chaque test réalisé. Lorsque nous évaluerons l'impact du module offline, nous utiliserons par abus de langage l'expression "module offline" pour désigner les performances obtenues lorsque le module offline est activé en plus du module online.

6.5.1 Offline sur les requêtes en avance

Dans cette section, le module offline est lancé une seule fois sur les requêtes des utilisateurs acceptées avant le début du service. Pour chaque instance, nous avons fait varier le temps maximal autorisé accordé au module offline. La figure 6.2 présente l'amélioration en terme de requêtes servies sur les deux groupes d'instances. On observe que l'amélioration en terme de requêtes servies tend à augmenter avec l'augmentation du temps maximal de recherche. Cette augmentation semble toutefois s'estomper au delà de 300 secondes (5 minutes), et ce pour les deux groupes. Augmenter le temps d'optimisation au-delà d'un certain seuil ne semble donc pas pertinent. Ceci est en partie expliqué par le fait que le temps disponible entre la dernière requête avant le début du service et le début du service limite le temps maximal de la recherche, quel que soit le paramètre donné. L'amélioration apportée reste cependant faible pour les deux groupes avec un maximum de 1.2%. L'intérêt de faire tourner le module uniquement sur les requêtes en avance est donc limité en terme de requêtes servies.



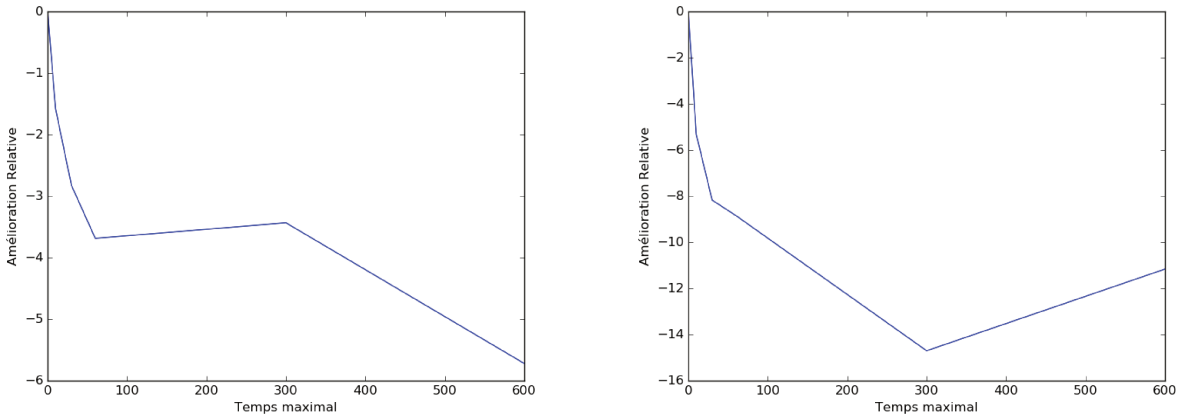
(a) Groupe A (b) Groupe B

Figure 6.2 – Amélioration relative du PSR par rapport au module online en fonction du temps maximal de recherche $(10, 30, 60, 300, 600$ secondes$)$

La figure 6.3a présente l'amélioration relative apportée sur la durée des tournées immédiatement après le lancement du module offline sur les requêtes en avance. On observe dans les deux groupes une plus grande amélioration lorsque le temps maximal augmente, la baisse de performances entre 300 et 600 secondes dans le groupe B étant probablement due à des effets stochastiques. Le passage de 60 à 300 secondes dans le groupe A (figure 6.3a) provoque une légère diminution de l'efficacité en terme de durée des véhicules mais une amélioration nette

1. https ://cython.org/

en terme de requêtes servies (figure 6.2a). De même, le passage de 30 à 60 secondes dans le groupe B provoque une amélioration en terme d'utilisation des véhicules (figure 6.3b) mais une diminution du nombre de requêtes servies (figure 6.2b). Il semble donc qu'il n'existe pas de corrélation systématique entre l'efficacité de l'optimisation sur les requêtes en avance et le nombre total de requêtes servies.



(a) Groupe A

(b) Groupe B

Figure 6.3 – Amélioration relative de la durée des véhicules obtenue après le lancer du module offline sur les requêtes en avance en fonction du temps maximal de recherche (10, 30, 60, 300, 600 secondes)

On observe également figure 6.2 que l'amélioration apportée par le module offline est plus grande sur le groupe A que pour le groupe B bien que l'optimisation soit plus efficace sur le groupe B (figure 6.2). Ceci est à mettre en relation avec la proportion de requêtes en avance de chaque groupe (cf section 3.3.2). Le groupe A contient en moyenne 66% de requêtes en avance tandis que le groupe B en contient 36%. De plus, les simulations montrent que 88% des requêtes acceptées par le module online sur les instances du groupe A sont des requêtes en avance contre seulement 44% sur le groupe B. Cela signifie que lorsque le module offline est lancé sur les requêtes en avance, la majorité des requêtes ont déjà été acceptées sur le groupe A. Il est possible que l'impact du module online sur le groupe B soit plus faible car bien que l'optimisation soit plus efficace, les impacts de son lancement "tôt" dans le service seront très vite annulés par les nombreuses requêtes encore à venir. Cela tend à montrer le besoin d'une procédure continue d'optimisation.

6.5.2 Offline en continu

Dans cette section, le module offline est lancé après chaque nouvelle requête arrivant au système. Le tableau 6.2 présente les résultats pour le groupe A. O représente le pourcentage de requêtes servies avec l'algorithme online, *Off* est le pourcentage de requêtes servies lorsque le module offline est activé en continu, *Imp* est l'amélioration relative du module offline sur l'algorithme online en terme de requêtes servies et *Imp-D* l'amélioration relative du module offline sur l'algorithme online en terme de durée des tournées.

Le module offline permet de servir un nombre plus important de requêtes dans 28 cas sur 30 et fournit une amélioration sur la durée des tournées dans 29 cas. Avec 13 véhicules, le module offline permet de servir plus de requêtes que le module online avec 14 véhicules sur 8 instances. Tout en économisant un véhicule, le module offline permet d'économiser en moyenne

Ve	12				13				14			
INS	O	Off	Imp	ImpD	O	O+R	Imp	ImpD	O	Off	Imp	ImpD
A_120	67.5	76.17	**12.84**	**-6.58**	74.17	78.75	**6.18**	**-9.38**	75.83	82.33	**8.57**	**-11.77**
A_145	53.10	57.93	**9.09**	**-7.51**	57.24	61.17	**6.87**	**-10.06**	62.06	66.21	**6.67**	**-8.78**
A_159	60.38	65.16	**7.92**	**-6.40**	66.04	68.68	**4.00**	**-6.67**	65.41	72.14	**10.29**	**-6.72**
A_177	42.37	45.93	**8.40**	0.58	44.63	48.31	**8.23**	**-6.63**	46.33	51.98	**12.20**	**-10.91**
A_216	43.52	42.78	-1.70	**-4.47**	44.44	47.18	**6.15**	**-5.48**	45.37	49.12	**8.27**	**-4.63**
A_233	51.07	57.60	**12.77**	**-2.53**	55.79	60.52	**8.46**	**-2.50**	60.94	62.58	**2.68**	**-5.24**
A_271	34.32	37.05	**7.96**	**-2.59**	36.53	39.41	**7.88**	**-7.90**	38.75	42.29	**9.14**	**-6.89**
A_288	34.03	33.99	-0.10	**-2.48**	34.72	37.53	**8.1**	**-4.21**	35.07	40.03	**14.16**	**-2.63**
A_288b	45.49	52.26	**14.89**	**-4.13**	54.86	55.56	**1.27**	**-3.32**	55.55	58.26	**4.87**	**-3.92**
A_360	26.39	29.44	**11.58**	**-2.60**	28.33	31.19	**10.1**	**-6.45**	30.27	33.81	**11.65**	**-5.12**
Avg.	45.82	49.83	**8.36**	**-3.87**	49.68	52.83	**6.72**	**-6.26**	51.56	55.87	**8.85**	**-6.66**

Tableau 6.2 – Performance du module offline en continue sur les instances du groupe A. Chaque instance est nommée G_N où G est le nom du groupe (A) et N le nombre de requêtes.

plus de 6% des coûts de transport. Des résultats similaires en terme de nombre de véhicules sont obtenus avec 12 véhicules : le module offline avec 12 véhicules permet de servir plus de requêtes sur 6 instances que le module online avec 13 véhicules sur 6 instances. Les meilleurs résultats sont obtenus avec 14 véhicules, où le module offline permet de servir 8.85% de requêtes en plus tout en réduisant les coûts de 6.66%. Le module offline permet donc d'apporter un gain très important à l'opérateur de transport même sur des instances très difficiles comme celles du groupe A.

Le tableau 6.3 présente les résultats pour le groupe B.

Ve	5				6				7			
INS	O	Off	Imp	ImpD	O	O+R	Imp	ImpD	O	Off	Imp	ImpD
B_200	84.5	87.75	**3.85**	**-8.64**	90.5	89.9	-0.66	**-16.56**	89.5	89.1	0.45	**-13.68**
B_301	82.06	86.35	**5.22**	**-13.43**	89.04	89.24	**0.22**	**-14.35**	91.03	90.93	-0.11	**-13.90**
B_398	86.93	85.13	-2.08	**-11.98**	88.94	88.49	-0.51	**-12.92**	87.44	90.75	**3.79**	**-9.92**
B_498	76.51	83.27	**8.85**	**-4.21**	85.94	88.61	**3.11**	**-7.97**	87.35	90.54	**3.66**	**-10.55**
B_607	72.32	79.41	**9.79**	**-7.05**	80.72	85.31	**5.68**	**-6.34**	84.51	88.54	**4.76**	**-7.40**
B_709	69.25	76.49	**10.45**	**-6.56**	80.82	81.50	**0.84**	**-6.99**	84.34	86.68	**2.78**	**-9.30**
B_802	67.83	72.91	**7.48**	**-6.59**	72.57	79.54	**9.60**	**-6.82**	78.05	84.83	**8.67**	**-9.38**
B_909	61.45	71.83	**16.88**	**-4.55**	73.68	79.86	**8.39**	**-5.11**	76.98	84.64	**9.94**	**-4.09**
B_1011	61.03	68.87	**12.85**	**-5.41**	70.23	76.41	**8.80**	**-4.79**	75.57	81.80	**8.25**	**-5.73**
Avg.	73.54	79.11	**8.14**	**-7.60**	81.38	84.32	**3.94**	**-9.1**	83.86	87.53	**4.59**	**-9.33**

Tableau 6.3 – Performance du module offline en continue sur les instances du groupe B. Chaque instance est nommée G_N où G est le nom du groupe (A) et N le nombre de requêtes.

Le module offline permet de servir un nombre plus important de requêtes dans 22 cas sur 27 et fournit une amélioration systématique sur la durée des tournées. Avec 6 véhicules, le module offline permet de servir plus de requêtes que le module online avec 7 véhicules sur 6 instances. En plus d'économiser un véhicule, le module offline permet de réduire le coût de transport de

presque 10% en moyenne, ce qui constitue une amélioration très significative. Des performances similaires sont également obtenues avec 7 véhicules. Les meilleurs résultats en terme de requêtes servies sont obtenus avec 5 véhicules, où l'amélioration en terme de requêtes servies est de 8.14% et la réduction des coûts de près de 7.60%. Quelque soit le nombre de véhicules, on observe que l'amélioration en terme de requêtes servies augmente et avoisine souvent les 10% lorsque le nombre de requêtes augmente. Ceci est lié au fait que plus le nombre de requêtes augmente, plus les choix myopes du module online sont loin de l'optimalité. Une heuristique d'optimisation comme l'ALNS a ainsi un apport plus conséquent. Le module offline est donc adapté pour traiter efficacement des instances de grande taille. Les instances du groupe B contenant 60% de requêtes en temps réel (section 3.3.2), les résultats montrent que le module offline est également capable de traiter efficacement des instances contenant une grande proportion de requêtes temps réel.

6.6 Conclusion

Dans ce chapitre, nous avons présenté en détails le module offline. L'heuristique principale du module offline est basée sur la métaheuristique ALNS. Nous avons montré comment les paramètres libres sont déterminés grâce à Irace, un logiciel de réglage des hyper-paramètres des algorithmes d'optimisation. Nous avons étudié les performances du module offline lorsqu'il est lancé en avance et après chaque requête. Nous avons vu que le lancement uniquement en avance du module offline apporte un gain limité en terme de requêtes servies, particulièrement lorsque la majorité des requêtes sont en temps réel. En revanche, le lancement en continu du module offline apporte un gain significatif à la fois en terme de requêtes servies et en terme de coûts de transport. Ce dernier permet ainsi d'économiser un véhicule tout en servant un nombre similaire de requêtes dans de nombreuses situations. Les coûts de transport sont de plus réduits de manière appréciable, entre 6 et 10% en moyenne. Le module offline a ainsi prouvé sa capacité à améliorer les performances sur des instances difficiles et/ou avec une forte proportion de requêtes temps réel. Enfin, les expérimentations sur le groupe B ont révélé le potentiel important du module offline sur des instances de grande taille.

Nous avons ainsi montré que le module offline est capable d'exploiter le temps libre durant le service pour produire des tournées de qualité et permettre de servir plus de requêtes. Les simulations ont été menées dans des conditions proches de la réalité, ce qui laisse présager que l'utilisation du module directement dans la plate-forme de Padam peut apporter des gains tout aussi significatifs.

Chapitre 7

Mise en oeuvre en contexte industriel

7.1 Introduction

En plus des contributions scientifiques présentées dans les chapitres précédents, l'objectif de cette thèse Cifre est également d'implémenter au coeur de la technologie de Padam les méthodes proposées pour permettre à l'entreprise de les utiliser sur ses différents territoires. Ce chapitre aborde l'intégration dans le progiciel des algorithmes présentés aux chapitres précédents et les difficultés rencontrées. Nous parlerons de mise en production (MEP) pour désigner le travail nécessaire pour passer d'un code fonctionnel et testable (prototype) à un code utilisé dans les services réels de Padam (production). Après avoir présenté brièvement l'environnement technologique de Padam, nous présenterons le travail de transformation des algorithmes présentés dans les chapitres 4, 5 et 6 en des modules informatiques flexibles et robustes.

7.2 Environnement technologique

L'équipe de développement est composée de 3 pôles : le Front-End, le Back-End et la R&D. Ces trois équipes travaillent en collaboration dans le cadre d'une méthode de management appelée Scrum : les tâches techniques à effectuer sont réparties sur des "sprints" de deux semaines avec des réunions de début et de mi-sprint pour planifier et suivre l'avancement des items prévus. Une réunion courte d'environ 15 minutes (le "stand-up") est tenue chaque midi pour permettre aux membres de l'équipe de situer leur travail dans l'avancement du sprint et de mettre en évidence d'éventuelles difficultés.

Chaque équipe est en charge d'une partie spécifique du produit : le Front-End (2 personnes) est en charge du développement des différentes applications mobiles et sites web. L'équipe Back-End (entre 3 et 5 personnes) gère en revanche le déploiement et la maintenance des différents serveurs, de l'application Web et de la base de données où sont stockées toutes les données concernant Padam et ses clients. La R&D (2 personnes) assure le développement et la maintenance des algorithmes. La base de code de l'équipe Back est contenue sur un service web d'hébergement et de gestion de développement de logiciels nommé Github, chaque développeur travaillant sur une copie locale. La collaboration entre les différents développeurs se fait via un système de gestion de version célèbre appelé Git. Le code de la R&D étant (pour le moment) inclus dans le code du Back-End, chaque membre de l'équipe R&D doit donc maîtriser Git et connaître à minima les fondamentaux de l'architecture logicielle de Django, le principal outil du Back-End.

7.2.1 Django

Django est un cadre de développement écrit en python permettant de déployer rapidement et en toute sécurité des applications web. L'image 7.1 présente un schéma descriptif de l'architecture MVT (Modèle-Vue-Template) à la base de Django.

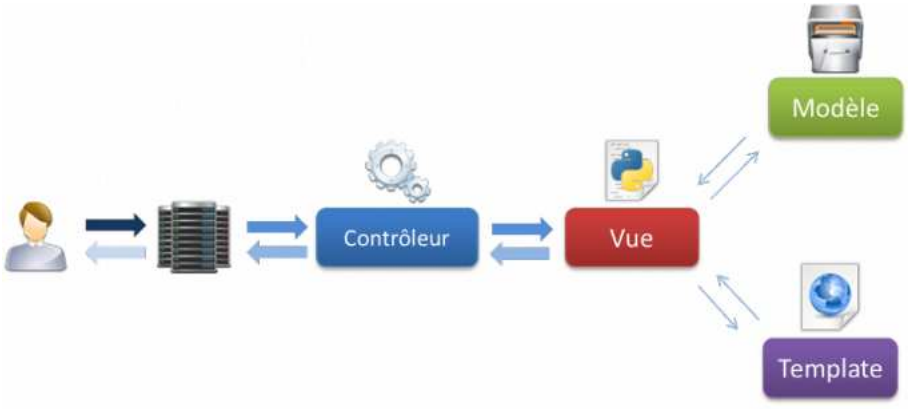


Figure 7.1 – Présentation générale de Django

Lorsque un utilisateur fait une requête depuis une application mobile ou un des sites web de réservation, une requête $HTTP$ est aussitôt envoyée aux serveurs mis en places par l'équipe Back-End pour le territoire. Django se charge ensuite via des règles de routage définies par les développeurs (partie "Controlleur") d'exécuter la "Vue" correspondante. Cette dernière récupère les données pertinentes des modèles pour effectuer sa tâche et génère ensuite une sortie en format HTML (via les gabarits ("Template")) ou simplement en un format d'échange de données standard comme le JSON. La sortie est ensuite transmise au contrôleur qui se charge de la renvoyer à l'utilisateur. L'ensemble des requêtes (URL) pouvant être traitées par l'application web est une Interface de Programmation Applicative (API). Le Front-End est ainsi indépendant du Back-end, la communication entre les deux étant assurée via cette API. Le modèle représente une information enregistrée dans une base de données (PostgreSQL, SQLite). Il s'agit d'une interface entre le code et la base de données qui permet d'accéder à l'information, de la modifier, d'en ajouter une nouvelle, de vérifier que celle-ci correspond bien aux critères (on parle d'intégrité de l'information), de la mettre à jour, etc... Toutes ces actions ne se font pas en SQL mais en python, la conversion étant assurée par un ORM (mapping objet-relationnel).

7.3 Développement d'un simulateur

La conception d'un outil permettant de simuler le système Padam pour étudier le comportement des algorithmes était un besoin mis en évidence dans l'entreprise avant le début de la thèse. Une première version du simulateur existait avant notre arrivée. Ce dernier avait cependant été développé dans l'urgence et n'avait pas été mis à jour depuis sa création. Les évolutions du code du Back-End n'avaient donc pas été prises en compte et le simulateur n'était plus en état de fonctionnement. De plus, il ne pouvait modéliser que très peu de cas d'usages. Ayant été écrit pour des situations spécifiques, il n'était pas possible de changer où d'ajouter rapidement des nouvelles fonctionnalités.

Le développement d'un simulateur opérationnel et modulaire fut donc notre première tâche dès l'arrivée dans l'entreprise. C'est ce simulateur que nous avons utilisé pour les expérimentations

présentées dans cette thèse. L'objectif principal de ce dernier est de permettre d'évaluer les réponses du système sous différents paramètres tels que la demande client, le nombre et la disposition des véhicules ou encore le paramétrage des algorithmes. Les résultats obtenus doivent être le plus proche possible de la réalité pour aboutir à des décisions pertinentes. Pour y parvenir en évitant la duplication de code, il nous a été nécessaire de relier le simulateur à certaines fonctions du code déjà existant dans le Back-End (permettant la création d'un service, le lancement de l'algorithme...). Cela nous a permis de mieux connaître et comprendre certains aspects fondamentaux du Back-End. Une utilisation des outils de gabarit de Django a aussi permis de produire en sortie du simulateur des fichiers de rendu lisibles pour interpréter les résultats. La figure 7.2 présente l'architecture globale du simulateur que nous avons conçue et implémentée.

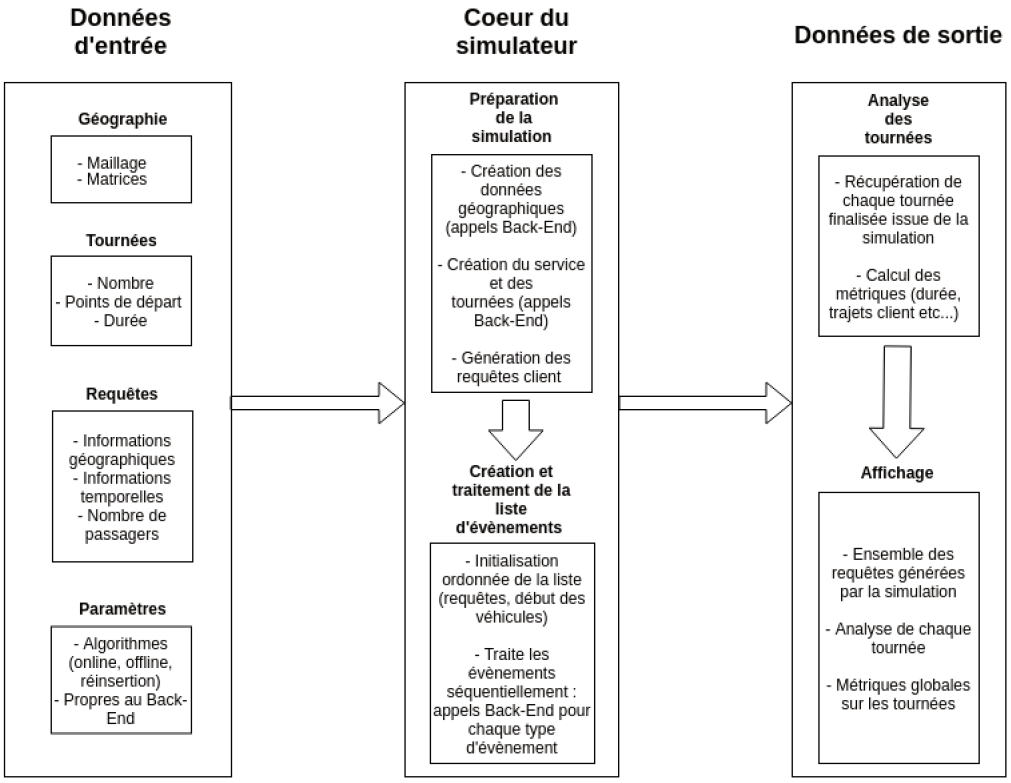


Figure 7.2 – Architecture générale du simulateur

La première composante du simulateur concerne les paramètres d'entrée nécessaires pour réaliser une simulation. Une partie importante des paramètres d'entrée concerne les requêtes client. L'utilisateur de la simulation peut choisir divers modes de génération des requêtes : statique ou aléatoire. En mode statique, un fichier de requêtes est envoyé qui sera ensuite lu pour générer les requêtes contenues. En mode aléatoire, l'utilisateur spécifie la loi d'apparition des requêtes sur les plans temporel (processus de poisson, loi normale etc...), géographique (uniforme, matrice origine/destination), du nombre de passagers et du type de la requête (orienté pickup ou dropoff). Contrairement au précédent simulateur, une grande variété de comportements est disponible.

Ces données sont ensuite envoyées au coeur du simulateur qui va utiliser les paramètres d'entrée pour créer les différentes composantes de la simulation (tournées, requêtes utilisateur etc...). Certaines étapes sont reliées directement à certaines fonctions du Back-End car elles nécessitent des fonctionnalités déjà présentes dans ce dernier. Nous avons assuré l'indépendance

des différentes étapes de la création de la simulation (géographiques, tournées, requêtes client) et nous avons conçu un code modulaire pour être en mesure de rajouter de nouveaux comportements si besoin, comme par exemple une nouvelle loi d'apparition des requêtes.

Une fois les principales composantes créées, les conditions initiales au déroulement de la simulation sont mises en place. Une simulation est considérée comme une suite d'évènements ordonnée suivant l'heure d'apparition représentant les évènements pouvant survenir durant un service réel : arrivée d'une réservation, arrivée et départ d'un véhicule à un noeud. L'état initial de la liste d'évènements est construit avec les requêtes générées à l'étape précédente et les évènements d'arrivée aux noeuds de départ des tournées. Les évènements concernant les déplacement des véhicules sont programmés dans la liste uniquement lorsque nécessaire. Par exemple, le départ du premier noeud d'une tournée est programmé lorsque l'évènement correspondant à l'arrivée au premier noeud (programmé à l'initialisation de la liste d'évènements) est réalisé. L'évènement correspondant à l'arrivée au deuxième noeud sera programmé dans la liste dès lors que l'évènement correspondant au départ du premier noeud aura été exécuté. Cette façon de faire est proche du comportement dans les services réels et permet de gérer beaucoup plus facilement les modifications faites aux tournées par l'arrivée de nouvelle requêtes. Chaque évènement de la simulation est exécuté par une fonction du Back-End correspondante pour assurer que le comportement de la simulation est le même que pour un service réel. La modélisation sous forme d'évènement nous a permis d'avoir une clarté et une modularité beaucoup plus grande en comparaison aux premières versions du simulateur.

La dernière étape de la simulation consiste à analyser les tournées une fois la simulation terminée et à afficher dans un format lisible et interprétable. Nous avons implémenté de nombreuses métriques en veillant à garder une modularité dans le code : chaque métrique est calculée par une fonction précise indépendante des autres. Il est donc facile de choisir les métriques à afficher et d'implémenter le calcul de nouvelles métriques si besoin.

Un autre aspect important de notre travail sur le simulateur a aussi été d'améliorer ses performances, principalement par l'utilisation de méthodes de parallélisme permettant de faire tourner un ensemble donné de simulations en parallèle sur différents coeurs.

Toutes ces améliorations apportées au cours des versions successives ont permis à l'entreprise de démarrer le produit PadamLab (cf section 2.1) au coeur duquel se situe le simulateur. De plus, les simulations de la thèse ont pleinement utilisé les différentes fonctionnalités du simulateur.

7.4 Architecture du nouveau système d'optimisation

Nous nous sommes aperçu très rapidement de l'importance de concevoir un code modulaire en extrayant des concepts clés et génériques dans du code réutilisable, sous forme de classe par exemple. Nous allons présenter dans cette section le code commun aux différents modules et les spécificités de chacun.

7.4.1 Code commun aux différents modules

Le premier module que nous avons développé est le module Offline. Nous avons tiré deux leçons fondamentales de son développement :

— L'importance de ne pas utiliser python et django dans le coeur des algorithmes
— L'importance de créer des briques de code réutilisables

Cython

La première version du module fut écrite en Python/Django. Ce choix nous a été suggéré par le fait que le premier module online alors existant était entièrement en Python/Django. Après avoir lancé les premières simulations, nous avons fait le constat suivant : les performances en terme de vitesse d'exécution étaient très mauvaises. Après une première étape de profilage, nous avons identifié que la communication avec la base de donnée effectuée via Django était le premier facteur limitant. Certaines parties fondamentales du code ont donc été réécrites dans le but d'y retirer tous les appels en base de données. Une amélioration appréciable des performances en est résultée. La performance globale restait cependant mauvaise. Le facteur limitant était désormais Python lui-même, à cause de sa caractéristique de typage dynamique qui en fait un language très lent en comparaison avec le C/C++. Après avoir testé différentes solutions, nous avons opté pour l'introduction d'un nouveau language/compilateur : Cython. Ce dernier permet d'interfacer facilement du python et du C/C++ et d'écrire directement du code atteignant des performances comparables au C tout en gardant une syntaxe proche de Python. Nous avons donc procédé à une réécriture complète du module, ce qui nous a permis de mettre en place deux choses importantes :

— Utiliser du Cython dans les parties du code critiques sur le plan de la performance.
— S'affranchir au plus de Django en créant nos propres modèles de données en Cython, différents des modèles de données Django (cf image 7.1) qui sont intrinsèquement liés à la base de données et nécessairement en pur Python. Une étape de conversion est nécessaire pour transformer les modèles Django en nos modèles Cython personnalisés. Cette transformation est faite une seule fois au lancement du module, la transformation inverse étant effectuée en sortie pour mettre à jour la base de données avec les solutions trouvées.

Ces deux étapes importantes nous ont permis d'obtenir un gain de performances allant de 1 à 2 ordres de magnitude (x10 - x100). La figure 7.3 schématise l'architecture mise en place. L'étape de transition permet à la fois de transformer les informations de la base de donnée (Modèles Django) vers les modèles Cython et d'envoyer les informations des objets Cython vers la base de données. Cette dernière étape est nécessaire pour mettre à jour la base de données après lancement des algorithmes.

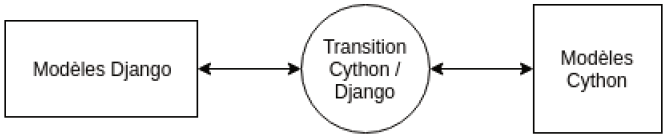


Figure 7.3 – Interaction Django/Cython

Modularité

Les modèles de données Cython présentés en figure 7.3 permettent de représenter les informations relatives aux tournées, requêtes des utilisateurs, noeuds du maillage etc... Ces modèles servent donc de base à tous les algorithmes implémentés et sont directement réutilisables pour l'implémentation des différents modules. De plus, les algorithmes développés dans cette thèse reposent sur les mêmes idées fondamentales : insertion et retrait de requêtes des tournées et calcul

d'objectif. La logique associée à ces fonctionnalités devrait donc être modulaire et réutilisable. La figure 7.4 résume les principales briques communes à tous les algorithmes. Les modules d'insertion, de retrait et de calcul d'objectifs exposent une ou plusieurs fonctions qui constituent les uniques points d'entrée depuis l'extérieur. L'implémentation interne de ces modules est donc indépendante du code extérieur. Nous avons utilisé des concepts classiques de Programmation Orienté Objet (POO) comme les interfaces, l'héritage et la composition pour atteindre un niveau de modularité suffisant. Le module de calcul d'objectifs expose par exemple une interface avec les méthodes nécessaires à implémenter permettant facilement l'ajout de nouvelles métriques sans modifier le code extérieur.

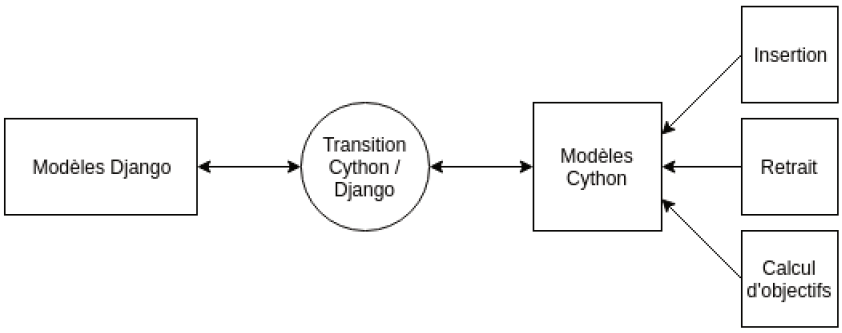


Figure 7.4 – Logique commune à tous les algorithmes

Les nouveaux contrats gagnés par Padam ont nécessité une organisation spécifique pour les modules d'insertion et de retrait. Certains territoires présentaient en effet des caractéristiques de service qui donnaient la possibilité d'adapter les algorithmes pour un traitement plus efficace. On peut citer à titre d'exemple le cas d'un service de rabattement vers une gare. Dans un tel service, les clients effectuent des trajets domicile-gare ou gare-domicile. Les heures de passage des trains étant déterminées à l'avance, l'heure de pickup ou de dropoff du client est triviale et dépendante de l'heure du train. Seule l'autre heure doit ainsi être calculée. Il est ainsi avantageux, à la fois d'un point de vue client et pour l'optimisation, d'adapter la façon d'insérer les requêtes dans un véhicule. Ce genre de spécificités n'étant pas rare, il nous a fallu retravailler la structure du code gérant l'insertion d'une requête dans un véhicule pour permettre d'ajouter facilement ce genre de spécificités tout en évitant la duplication de code et en maintenant une bonne lisibilité du code. La figure 7.5 illustre cette idée, valable tant pour le module d'insertion que de retrait dont il est question à la figure 7.4 : les entrées et sorties sont les mêmes quel que soit l'algorithme les utilisant, la distinction étant invisible pour l'extérieur. Il est donc possible d'ajouter des nouvelles façons d'insérer et de retirer les requêtes sans avoir à modifier le code extérieur et la logique haut-niveau des algorithmes. De plus, utiliser cette architecture permet de pouvoir utiliser n'importe quel algorithme sur n'importe quel territoire en étant sûr que la façon de modifier les tournées (via insertion et retrait) est identique.

7.4.2 Module Offline

Le module offline basé sur l'approche ALNS que nous avons proposée dans le chapitre 6 est le premier algorithme que nous avons implémenté dans la thèse, à la suite d'une première phase de bibliographie.

La figure 7.6 présente un résumé de l'architecture globale du module offline. Ce dernier utilise pleinement les modules bas-niveau présentés en figure 7.4 (les interactions cython/django ne

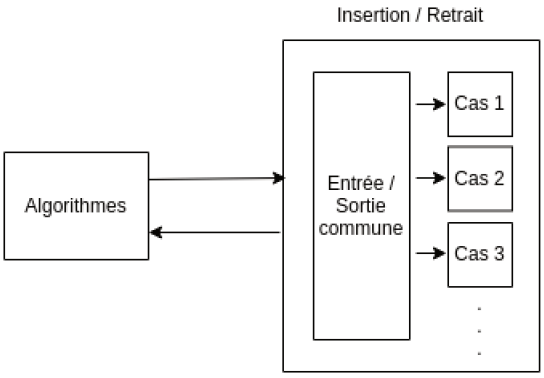


Figure 7.5 – Architecture des modules d'insertion/retrait

sont pas reproduites par soucis de clarté). Le module de "Multi-insertion" implémente toutes les fonctionnalités utilisées par les opérateurs de perturbation de l'ALNS (6.3.4) pour l'insertion séquentielle d'une liste de requêtes (mise à jour des insertions faisables, mise à jour de l'objectif au fur et à mesure des insertions etc...). Cette brique de code est utilisée par tous les opérateurs de perturbation et permet d'ajouter rapidement des opérateurs en se concentrant uniquement sur la logique propre à l'opérateur. De la même manière, le module de Multi-Retrait permet de retirer une liste de requêtes des tournées en mettant à jour la fonction objective. L'ALNS utilise ensuite ces deux modules pour modifier les tournées. Bien que l'ALNS soit actuellement l'heuristique utilisée, l'architecture du code permet d'implémenter de nouvelles heuristiques sans avoir à ré-implémenter toute la logique propre à Padam.

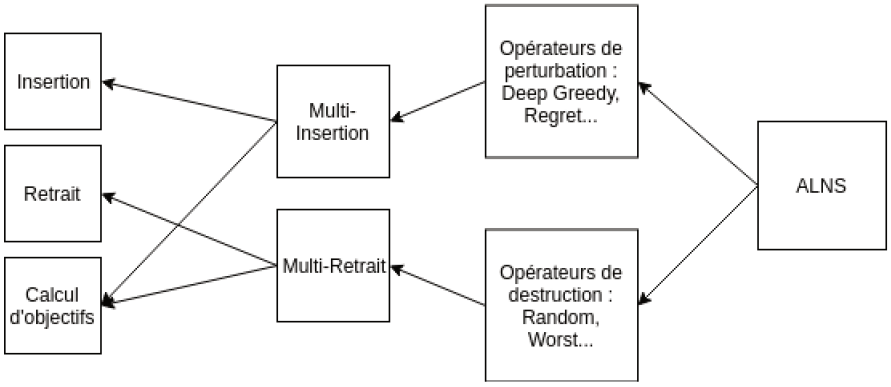


Figure 7.6 – Résumé de l'architecture globale du module offline

Tout le travail effectué pour rendre le module offline utilisable facilement depuis le code du Back-End (et dans les simulations) a grandement facilité sa mise en production. Un autre facteur cependant était indispensable pour sa mise en production : l'écriture de tests automatiques. Ces derniers permettent en effet de détecter d'éventuels bugs en testant les diverses fonctionnalités d'un code. Ces tests sont lancés automatiquement dès lors qu'un code doit être mis en production pour assurer une bonne qualité du produit. Nous nous sommes donc attelés à l'écriture de tests unitaires et d'intégrations pour le module offline. Une fois ces étapes réalisées, le module offline a pu être utilisé sur des services réels, mais uniquement sur les requêtes en avance d'un service. Ainsi, quelques dizaines de minutes avant le début d'un service, l'offline est lancé quelques minutes pour optimiser les tournées déjà existantes. Cette mise en production

s'est réalisée en deux étapes principales :

— Le module offline a d'abord été lancé sans impacter la base de données, en fonctionnement fantôme. Cela signifie que les résultats trouvés par le module n'étaient pas répercutés sur le service associé mais étaient envoyés par mail à plusieurs membres de l'équipe pour permettre une première vérification manuelle.
— Une fois les vérifications faites (sur plusieurs semaines), le module a été activé en étant autorisé à modifier la base de données.

L'utilisation du module en mode semi-offline, c'est-à-dire en continu durant le service n'est pour le moment pas possible. En effet, de nombreux évènements modifiant les tournées (mise à jour des horaires suite au traffic par exemple) ont lieu en temps réel et il est nécessaire que le module en ait connaissance pour travailler efficacement. Une première solution serait de lancer le module offline à des intervalles réguliers et mettre à jour la base de données si les tournées n'ont pas changé entre-temps. Cette approche n'est toutefois pas la meilleure si la base de données est modifiée très souvent car le module n'aura jamais l'opportunité de modifier la base de données. Il est plus avantageux d'avoir une communication asynchrone entre le Back-End et le module offline pour permettre au module offline d'être informé de toutes les modifications de la base de données. Des réflexions sont en cours pour faire évoluer l'architecture tant du Back-End que du module offline et permettre cette avancée.

7.4.3 Module Online

Comme expliqué dans le chapitre 2, une version initiale du module Online était le seul module d'optimisation chez Padam. En plus des inconvénients de ce module décrits dans le chapitre 2, ce module original était écrit en Python/Django et souffrait donc d'importantes lacunes en terme de performances, tout comme la première version du module offline. Nous avons proposé une nouvelle version du module Online avec plusieurs améliorations (cf chapitre 4), et nous avons donc décidé de le réécrire en Cython en le séparant à son tour des interactions avec la base de données, comme dans le cas du module offline. La figure 7.7 présente l'architecture globale du module online. Les briques d'insertion et de calcul des objectifs de la figure 7.4 sont réutilisées (les interactions cython/django ne sont pas reproduites par soucis de clarté). Le moteur d'insertion est décrit aux sections 4.3.1 et 4.3.2 et le module de filtre aux sections 4.3.3, 4.3.4 et 4.3.5. Le module de filtre est conçu en utilisant des interfaces pour permettre d'ajouter facilement de nouveaux filtres en fonction du besoin. Ces deux modules forment l'ensemble de l'heuristique d'insertion décrite en section 4.3. Elle est appelée par le code gérant la transformation en propositions pour l'utilisateur et l'interaction avec le Back-End et également par la procédure de validation.

Une fois l'algorithme présenté au chapitre 4 implémenté, plusieurs étapes étaient encore indispensables avant sa mise en production. Il était nécessaire d'écrire de nouveaux tests automatiques et d'adapter les tests d'intégration déjà existant pour l'ancien module. Il était de plus vital d'intégrer un système de logs à l'intérieur du module. Ces derniers sont en effet des sources d'informations indispensables lorsqu'un bug survient lors d'une recherche client en production. Une fois le code prêt, la mise en production s'est réalisée de la manière suivante :

— Le nouveau module online a d'abord été lancé sans impacter la base de données, en fonctionnement fantôme. Cela signifie qu'à chaque nouvelle requête, le nouveau module était appelé après le module existant mais ses résultats n'impactaient pas la base de

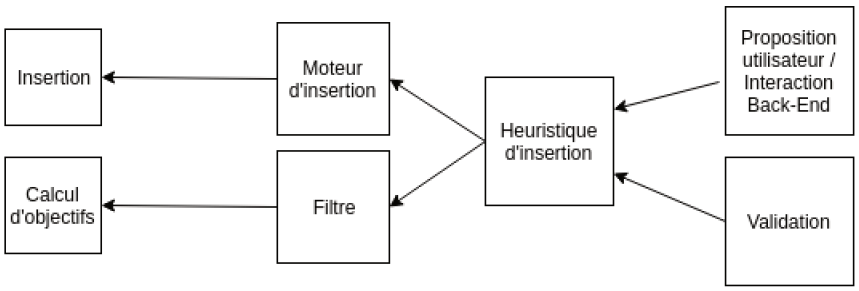


Figure 7.7 – Résumé de l'architecture globale du module online

données. Ils étaient écrit dans un fichier séparé pour être analysé en vue de repérer d'éventuelles erreurs.

— Une fois les vérifications validées (sur environ deux semaines), le nouveau module online est devenu effectif et l'ancien module a été supprimé.

Une fois la mise en production effective, nous avons assuré le maintien et le support en corrigeant des erreurs ponctuelles survenant lors de certaines requêtes. Un des problèmes majeurs auxquels il nous a fallu faire face est liée à la performance. En effet, le module online doit fournir des réponses rapides au client, au plus 2 secondes. Ce temps est significativement augmenté lorsque la taille du service (nombre de véhicules et de réservations) augmente. De nombreuses étapes de profilage ont été nécessaires pour identifier les goulots d'étranglement et les corriger. Un des éléments les plus difficiles à corriger fut la partie de transformation des modèles Django en modèles Cython (cf figure 7.3). Cette dernière est fortement ralentie par les appels en base de données qui sont évidemment incontournables. Nos premières optimisations ont consisté à minimiser le nombre d'appels en utilisant les optimisations fournies par l'ORM de Django. Les performances étaient bonnes mais se sont révélées insuffisantes lorsqu'un nouveau territoire avec plusieurs centaines de tournées a été ouvert. Il a été nécessaire de restructurer le code une fois de plus en utilisant certaines fonctionnalités de l'ORM Django pour effectuer les appels en base de données les plus légers possibles et pour permettre que la plus grande partie du traitement des données se fasse en Cython et non via Django (implémenté en Python). Ceci nous a permis de descendre en-dessous du seuil de 0.25s de temps de transformation des modèles pour une base de données contenant plus de 300 tournées.

7.4.4 Module de Réinsertion

Une fois le module Online fonctionnel et mis en production, nous avons pu nous concentrer sur le développement du module de Réinsertion présenté au chapitre 5. La figure 7.8 présente l'architecture globale du module de réinsertion. Une partie du code se charge de l'interaction avec l'online et décide de l'heuristique à appeler. L'heuristique HDR (cf section 5.3) réutilise les briques de code développées pour les opérateurs de l'ALNS et les heuristiques HG/HGA (cf sections 5.5 et 5.6) sont basées sur un module de calcul du graphe qui utilise pleinement les modules d'insertion, de retrait et de calculs d'objectifs.

Une grande partie du développement associé au module a consisté à adapter le code existant (Back-End et Online) pour intégrer la réinsertion au processus de réservation client (partie "Interaction Online" sur la figure 7.8). Ceci nous à permis de l'utiliser naturellement dans les simulations. Cependant, le module de Réinsertion est le seul qui ne soit pas encore en production. Les tâches principales à réaliser pour y parvenir concernent la mise en place d'une

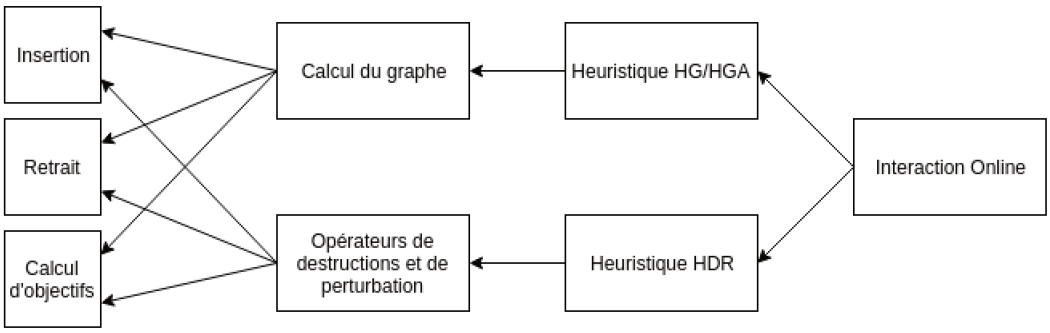


Figure 7.8 – Résumé de l'architecture globale du module de réinsertion

procédure de validation semblable à celle du module online (cf section 4.2.4) et le développement d'un système de logs. L'objectif de la procédure de validation est de vérifier qu'une proposition faite au client est encore valable lorsque le client décide de valider, ce qui peut avoir lieu plusieurs dizaines de secondes après la recherche. Les simulations ne permettant pas actuellement de tester ce genre de comportement, il nous faudra améliorer le simulateur et procéder à des tests sur un service réel. Une façon possible serait d'activer en fantôme le module de Réinsertion et d'étudier le comportement qui en résulte.

7.5 Conclusion

Dans ce chapitre, nous avons présenté le travail mené pour que les contributions de la thèse soient utilisables directement par l'entreprise. Notre travail en contexte industriel nous a permis de découvrir les difficultés liés à la mise en production d'un algorithme ainsi que le temps très important nécessaire à sa maintenance et son évolution. Des critères de qualité et de maintenance du code sont à considérer sérieusement pour établir un code pérenne. De plus, certains aspects comme la gestion des logs et l'écriture de tests automatiques sont à prendre en compte. Bien que souvent négligés, ces aspects sont fondamentaux pour développer un logiciel fiable, robuste et facilement analysable lorsque des problèmes surgissent.

Le travail fourni a permis de rendre utilisable par l'entreprise une partie importante des contributions réalisées durant la thèse. Parmi les défis techniques importants encore à surmonter se trouvent notamment la mise en production continue du module offline. Une solution pour ce problème serait d'instaurer un système de cache mis à jour à chaque modification des objets de la base de données. Le module offline aurait accès à ce cache en lecture et aurait accès en temps réel à chaque modification réalisée sur la base de données. Un autre défi à relever tient à la mise en production du module de réinsertion, qui nécessite un travail sur une procédure de validation efficace et sur le système de logs. Enfin, nous travaillons avec d'autres membres de l'équipe à la possibilité de séparer le code de la R&D et du Back-End. Les algorithmes de la R&D seraient exécutés sur un serveur séparé et la communication avec le Back serait assurée via une API. L'avantage d'une telle infrastructure est de permettre aux deux équipes de travailler en indépendance et de pouvoir dimensionner les serveurs de la R&D et du Back-End en fonction de leurs besoins respectifs.

Chapitre 8

Conclusion générale et perspectives

8.1 Conclusion

Le transport en commun à la demande est une composante majeure de la mobilité de demain. Il offre à l'utilisateur une grande flexibilité à un tarif compétitif en proposant une alternative solide aux systèmes de transport en commun classiques. Cette thèse CIFRE a permis de développer des outils d'optimisation pour Padam, une start-up parisienne proposant un système de transport en commun à la demande innovant. Nous nous sommes intéressés aux problèmes liés à l'optimisation dynamique des tournées de véhicules, modélisée dans la littérature scientifique par le DARP dynamique. L'état de l'art que nous avons effectué a clairement mis en évidence les manques vis à vis de ce problème et a fait l'objet d'une présentation à une conférence nationale de recherche opérationnelle ([Vallée et al., 2016a]). Pour pallier ces spécificités du problème Padam, nous avons développé plusieurs modules d'optimisation complémentaires directement intégrables dans la technologie de l'entreprise. Nous avons également mené à bien le développement d'un simulateur qui permet de tester et d'ajuster les algorithmes d'optimisation proposés. De nombreuses expérimentations basées sur des données réelles de l'entreprise ont ainsi pu être réalisées. Ce simulateur fait désormais partie intégrante de l'offre commerciale de l'entreprise.

Nous avons abordé le problème du traitement temps réel des requêtes, qui est un sujet crucial pour Padam. Nous avons présenté un module online qui constitue le premier levier d'optimisation et est actuellement utilisé par l'entreprise sur des services réels. L'heuristique développée permet de créer des tournées de qualité tout en respectant les contraintes horaires serrées des utilisateurs. Elle se distingue des heuristiques de la littérature par son aspect multi-horaire et par sa capacité à intégrer les noeuds de prise en charge et de dépose dans l'optimisation. Cette dernière fonctionnalité s'est montrée très efficace en permettant de servir plus de requêtes avec des coûts moindres.

Nous avons également mené une étude sur la réinsertion temps réel des requêtes rejetées par le module online, qui est un sujet quasi-absent de la littérature scientifique. Nous avons proposé deux nouvelles heuristiques sur lesquelles nous avons réalisé d'intenses expérimentations. Nous avons démontré l'intérêt de ces heuristiques sur les performances du système, notamment grâce à leur capacité à économiser l'usage d'un véhicule dans de nombreuses situations. Ces deux heuristiques sont de plus utilisables en temps réel sans perturber de manière significative le temps de réponse au client. Ces caractéristiques constituent un avantage important pour l'entreprise et les rendent directement applicables sur des services réels. Le travail mené a fait

l'objet d'une publication dans une conférence internationale ([Vallée et al., 2019]).

Un autre aspect de notre travail a consisté à développer un module offline permettant d'optimiser les tournées en continu. Ce module est basé sur une méta-heuristique à grand voisinage (ALNS), qui n'a jamais été utilisée dans le cadre du DARP dynamique. Nous avons montré l'impact significatif du module offline qui permet de servir plus de requêtes tout en réduisant les coûts de transport. Il permet ainsi d'économiser un véhicule dans de nombreuses situations. Le travail mené a fait l'objet d'une présentation à une conférence internationale ([Vallée et al., 2016b]) ainsi qu'une publication dans une autre conférence internationale ([Vallée et al., 2017]).

Enfin, nous avons montré l'architecture globale que nous avons implémentée pour le simulateur et les 3 modules d'optimisation. Nous avons exposé les difficultés rencontrées et les points important à prendre en compte pour transformer les algorithmes présentés en des modules d'optimisation robustes, flexibles et utilisables par l'entreprise. La plupart des travaux réalisés durant cette thèse sont actuellement utilisés par Padam sur des services réels.

8.2 Perspectives

Les sujets abordés dans cette thèse sont d'actualité et peu traités dans la littérature scientifique. De nombreuses perspectives de recherche sont donc envisageables. De plus, la mise en production d'une partie du travail est encore à réaliser et soulève d'important défis opérationnels.

8.2.1 Perspectives scientifiques

— Le fonctionnement actuel de l'heuristique de réinsertion à voisinage large présentée dans le chapitre 5 implique que de nombreux mouvements inutiles sont tentés, ce qui engendre nécessairement une perte de temps. Une amélioration importante consiste à apprendre les mouvements pertinents pour déterminer rapidement des solutions voisines de qualité. Une possibilité d'apprentissage serait d'utiliser des réseaux de neurones (profonds/convolutifs) pour représenter une solution et les mouvements possibles. L'entraînement serait ensuite réalisé en essayant de nombreux mouvements sur des solutions générées aléatoirement pour apprendre au réseau à reconnaître ce qui définit un mouvement pertinent pour une solution donnée.

— L'heuristique du module online présentée au chapitre 4 permet d'offrir plusieurs propositions à l'utilisateur en fonction de l'heure demandée. Actuellement le cadencement des propositions est statique, dans la mesure où les insertions sont groupées pour être comparées dans des intervalles indépendants des caractéristiques de la recherche en question. Une piste d'amélioration serait de créer un cadencement qui adapte les intervalles en fonction de la répartition des insertions faisables trouvées.

— L'objectif principalement utilisé durant cette thèse est la durée totale des tournées. Bien que pertinent, cet objectif ne prend pas en compte la distribution stochastique des futures requêtes et ne fournit donc aucun levier d'adaptation pour prendre en compte les données historiques. Une potentielle amélioration consiste à concevoir un objectif qui cherche à maximiser la probabilité d'insertion des futures clients selon la distribution de probabilité sous-jacente.

— Les algorithmes présentés durant cette thèse n'optimisent à chaque fois qu'un seul objectif. Il peut-être bénéfique au système d'utiliser une fonction multi-objectif en intégrant une ou plusieurs autres métriques en parallèle dans le processus d'optimisation.

8.2.2 Perspectives opérationnelles

— Le module de réinsertion n'est actuellement pas en production. Le développement d'une procédure de validation efficace est nécessaire.

— L'utilisation du module offline en continu n'est actuellement pas possible en production. Pour y parvenir, il est nécessaire que le module offline ait une communication continue avec la base de données pour être informé de la moindre modification des modèles. Une façon d'y parvenir serait d'utiliser certaines fonctionnalités de django pour répercuter toutes les modifications des modèles vers un cache en mémoire auquel le module offline serait connecté. Ce dernier serait ainsi en mesure d'intégrer les modifications de la base de données directement à l'intérieur de sa recherche et de pouvoir envoyer les nouvelles meilleures solutions trouvées en base de données sans risque d'obsolescence de l'information.

— Le code de la R&D et du Back-End sont actuellement sur le même répertoire et sur le même serveur de production. Cette architecture présente de nombreux inconvénients. Une perspective importante d'évolution technique consiste à séparer les deux entités, en établissant une communication non plus par le code lui-même mais par une API.

Bibliographie

[Atahran et al., 2014] Atahran, A., Lenté, C., and T'kindt, V. (2014). A multicriteria dial-a-ride problem with an ecological measure and heterogeneous vehicles. *Journal of Multi-Criteria Decision Analysis*, 21(5-6) :279–298.

[Attanasio et al., 2004] Attanasio, A., Cordeau, J.-F., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3) :377–387.

[Baker and Ayechew, 2003] Baker, B. M. and Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5) :787–800.

[Balas and Simonetti, 2001] Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted tsps : A computational study. *INFORMS journal on Computing*, 13(1) :56–75.

[Beaudry et al., 2010] Beaudry, A., Laporte, G., Melo, T., and Nickel, S. (2010). Dynamic transportation of patients in hospitals. *OR spectrum*, 32(1) :77–107.

[Bent and Van Hentenryck, 2004] Bent, R. W. and Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6) :977–987.

[Braekers et al., 2014] Braekers, K., Caris, A., and Janssens, G. K. (2014). Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B : Methodological*, 67 :166–186.

[Braekers and Kovacs, 2016] Braekers, K. and Kovacs, A. A. (2016). A multi-period dial-a-ride problem with driver consistency. *Transportation Research Part B : Methodological*, 94 :355–377.

[Chassaing et al., 2016] Chassaing, M., Duhamel, C., and Lacomme, P. (2016). An els-based approach with dynamic probabilities management in local search for the dial-a-ride problem. *Engineering Applications of Artificial Intelligence*, 48 :119–133.

[Chevrier et al., 2012] Chevrier, R., Liefooghe, A., Jourdan, L., and Dhaenens, C. (2012). Solving a dial-a-ride problem with a hybrid evolutionary multi-objective approach : Application to demand responsive transport. *Applied Soft Computing*, 12(4) :1247–1258.

[Cordeau, 2006] Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3) :573–586.

[Cordeau and Laporte, 2003] Cordeau, J.-F. and Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B : Methodological*, 37(6) :579–594.

[Cordeau and Laporte, 2007] Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem : models and algorithms. *Annals of operations research*, 153(1) :29–46.

[Coslovich et al., 2006] Coslovich, L., Pesenti, R., and Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3) :1605–1615.

[Cubillos et al., 2009] Cubillos, C., Urra, E., and Rodríguez, N. (2009). Application of genetic algorithms for the darptw problem. *International Journal of Computers Communications & Control*, 4(2) :127–136.

[Detti et al., 2017] Detti, P., Papalini, F., and de Lara, G. Z. M. (2017). A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega*, 70 :1–14.

[Diana and Dessouky, 2004] Diana, M. and Dessouky, M. M. (2004). A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B : Methodological*, 38(6) :539–557.

[Feo and Resende, 1995] Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2) :109–133.

[Garaix et al., 2010] Garaix, T., Artigues, C., Feillet, D., and Josselin, D. (2010). Vehicle routing problems with alternative paths : An application to on-demand transportation. *European Journal of Operational Research*, 204(1) :62–75.

[Garaix et al., 2011] Garaix, T., Artigues, C., Feillet, D., and Josselin, D. (2011). Optimization of occupancy rate in dial-a-ride problems via linear fractional column generation. *Computers & Operations Research*, 38(10) :1435–1442.

[Gendreau et al., 2006] Gendreau, M., Guertin, F., Potvin, J.-Y., and Séguin, R. (2006). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C : Emerging Technologies*, 14(3) :157–174.

[Glover, 1992] Glover, F. (1992). New ejection chain and alternating path methods for traveling salesman problems. In *Computer science and operations research*, pages 491–509. Elsevier.

[Gschwind and Drexl, 2019] Gschwind, T. and Drexl, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*.

[Gschwind and Irnich, 2014] Gschwind, T. and Irnich, S. (2014). Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, 49(2) :335–354.

[Guerriero et al., 2013] Guerriero, F., Bruni, M. E., and Greco, F. (2013). A hybrid greedy randomized adaptive search heuristic to solve the dial-a-ride problem. *Asia-Pacific Journal of Operational Research*, 30(01) :1250046.

[Häll et al., 2012] Häll, C. H., Högberg, M., and Lundgren, J. T. (2012). A modeling system for simulation of dial-a-ride services. *Public Transport*, 4(1) :17–37.

[Häll et al., 2015] Häll, C. H., Lundgren, J. T., and Voß, S. (2015). Evaluating the performance of a dial-a-ride service using simulation. *Public Transport*, 7(2) :139–157.

[Häll and Peterson, 2013] Häll, C. H. and Peterson, A. (2013). Improving paratransit scheduling using ruin and recreate methods. *Transportation Planning and Technology*, 36(4) :377–393.

[Ho et al., 2018] Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems : Literature review and recent developments. *Transportation Research Part B : Methodological*, 111 :395–421.

[Hu and Chang, 2015] Hu, T.-Y. and Chang, C.-P. (2015). A revised branch-and-price algorithm for dial-a-ride problems with the consideration of time-dependent travel cost. *Journal of Advanced Transportation*, 49(6) :700–723.

[Ilani et al., 2014] Ilani, H., Shufan, E., Grinshpoun, T., Belulu, A., and Fainberg, A. (2014). A reduction approach to the two-campus transport problem. *Journal of Scheduling*, 17(6) :587–599.

[Jaw et al., 1986] Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., and Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B : Methodological*, 20(3) :243–257.

[Jorgensen et al., 2007] Jorgensen, R. M., Larsen, J., and Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. *Journal of the operational research society*, 58(10) :1321–1331.

[Kirchler and Calvo, 2013] Kirchler, D. and Calvo, R. W. (2013). A granular tabu search algorithm for the dial-a-ride problem. *Transportation Research Part B : Methodological*, 56 :120–135.

[Lehuédé et al., 2014] Lehuédé, F., Masson, R., Parragh, S. N., Péton, O., and Tricoire, F. (2014). A multi-criteria large neighbourhood search for the transportation of disabled people. *Journal of the Operational Research Society*, 65(7) :983–1000.

[Li et al., 2016] Li, Y., Chen, H., and Prins, C. (2016). Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, 252(1) :27–38.

[Lim and Zhang, 2007] Lim, A. and Zhang, X. (2007). A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 19(3) :443–457.

[Lim et al., 2016] Lim, A., Zhang, Z., and Qin, H. (2016). Pickup and delivery service with manpower planning in hong kong public hospitals. *Transportation Science*, 51(2) :688–705.

[Liu et al., 2015] Liu, M., Luo, Z., and Lim, A. (2015). A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B : Methodological*, 81 :267–288.

[Lois and Ziliaskopoulos, 2017] Lois, A. and Ziliaskopoulos, A. (2017). Online algorithm for dynamic dial a ride problem and its metrics. *Transportation research procedia*, 24 :377–384.

[López-Ibáñez et al., 2016a] López-Ibáñez, M., Cáceres, L. P., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2016a). The irace package : User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004*.

[López-Ibáñez et al., 2016b] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016b). The irace package : Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3 :43–58.

[Luo and Schonfeld, 2007] Luo, Y. and Schonfeld, P. (2007). A rejected-reinsertion heuristic for the static dial-a-ride problem. *Transportation Research Part B : Methodological*, 41(7) :736–755.

[Luo and Schonfeld, 2011] Luo, Y. and Schonfeld, P. (2011). Online rejected-reinsertion heuristics for dynamic multivehicle dial-a-ride problem. *Transportation Research Record : Journal of the Transportation Research Board*, pages 59–67.

[Maalouf et al., 2014] Maalouf, M., MacKenzie, C. A., Radakrishnan, S., and Court, M. (2014). A new fuzzy logic approach to capacitated dynamic dial-a-ride problem. *Fuzzy Sets and Systems*, 255 :30–40.

[Marković et al., 2015] Marković, N., Nair, R., Schonfeld, P., Miller-Hooks, E., and Mohebbi, M. (2015). Optimizing dial-a-ride services in maryland : benefits of computerized routing and scheduling. *Transportation Research Part C : Emerging Technologies*, 55 :156–165.

[Masmoudi et al., 2017] Masmoudi, M. A., Braekers, K., Masmoudi, M., and Dammak, A. (2017). A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & operations research*, 81 :1–13.

[Masmoudi et al., 2016] Masmoudi, M. A., Hosny, M., Braekers, K., and Dammak, A. (2016). Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research Part E : Logistics and Transportation Review*, 96 :60–80.

[Masson et al., 2013] Masson, R., Lehuédé, F., and Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3) :344–355.

[Masson et al., 2014] Masson, R., Lehuédé, F., and Péton, O. (2014). The dial-a-ride problem with transfers. *Computers & Operations Research*, 41 :12–23.

[Molenbruch et al., 2017] Molenbruch, Y., Braekers, K., Caris, A., and Berghe, G. V. (2017). Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation. *Computers & Operations Research*, 77 :58–71.

[Muelas et al., 2015] Muelas, S., LaTorre, A., and Pena, J.-M. (2015). A distributed vns algorithm for optimizing dial-a-ride problems in large-scale scenarios. *Transportation Research Part C : Emerging Technologies*, 54 :110–130.

[Núñez et al., 2014] Núñez, A., Cortés, C. E., Sáez, D., De Schutter, B., and Gendreau, M. (2014). Multiobjective model predictive control for dynamic pickup and delivery problems. *Control Engineering Practice*, 32 :73–86.

[Paquette et al., 2013] Paquette, J., Cordeau, J.-F., Laporte, G., and Pascoal, M. M. (2013). Combining multicriteria analysis and tabu search for dial-a-ride problems. *Transportation Research Part B : Methodological*, 52 :1–16.

[Parragh, 2011] Parragh, S. N. (2011). Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C : Emerging Technologies*, 19(5) :912–930.

[Parragh et al., 2010] Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6) :1129–1138.

[Parragh et al., 2009] Parragh, S. N., Doerner, K. F., Hartl, R. F., and Gandibleux, X. (2009). A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks : An International Journal*, 54(4) :227–242.

[Parragh et al., 2014] Parragh, S. N., Pinho de Sousa, J., and Almada-Lobo, B. (2014). The dial-a-ride problem with split requests and profits. *Transportation Science*, 49(2) :311–334.

[Pimenta et al., 2017] Pimenta, V., Quilliot, A., Toussaint, H., and Vigo, D. (2017). Models and algorithms for reliability-oriented dial-a-ride with autonomous electric vehicles. *European Journal of Operational Research*, 257(2) :601–613.

[Psaraftis, 1980] Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2) :130–154.

[Qu and Bard, 2013] Qu, Y. and Bard, J. F. (2013). The heterogeneous pickup and delivery problem with configurable vehicle capacity. *Transportation Research Part C : Emerging Technologies*, 32 :1–20.

[Qu and Bard, 2014] Qu, Y. and Bard, J. F. (2014). A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity. *Transportation Science*, 49(2) :254–270.

[Rachid, 2010] Rachid, M. H. (2010). *Les problèmes de tournées de véhicules en planification industrielle : classification et comparaison d'opérateurs évolutionnaires*. PhD thesis, Université de Franche-Comté.

[Rego, 2001] Rego, C. (2001). Node-ejection chains for the vehicle routing problem : Sequential and parallel algorithms. *Parallel Computing*, 27(3) :201–222.

[Ritzinger et al., 2016] Ritzinger, U., Puchinger, J., and Hartl, R. F. (2016). Dynamic programming based metaheuristics for the dial-a-ride problem. *Annals of Operations Research*, 236(2) :341–358.

[Ropke et al., 2007] Ropke, S., Cordeau, J.-F., and Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks : An International Journal*, 49(4) :258–272.

[Ropke and Pisinger, 2006] Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4) :455–472.

[Santos and Xavier, 2013] Santos, D. O. and Xavier, E. C. (2013). Dynamic taxi and ridesharing : A framework and heuristics for the optimization problem. In *IJCAI*, volume 13, pages 2885–2891.

[Santos and Xavier, 2015] Santos, D. O. and Xavier, E. C. (2015). Taxi and ride sharing : A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19) :6728–6737.

[Sayarshad and Chow, 2015] Sayarshad, H. R. and Chow, J. Y. (2015). A scalable non-myopic dynamic dial-a-ride and pricing problem. *Transportation Research Part B : Methodological*, 81 :539–554.

[Schilde et al., 2011] Schilde, M., Doerner, K. F., and Hartl, R. F. (2011). Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & operations research*, 38(12) :1719–1730.

[Schilde et al., 2014] Schilde, M., Doerner, K. F., and Hartl, R. F. (2014). Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. *European journal of operational research*, 238(1) :18–30.

[Schönberger, 2017] Schönberger, J. (2017). Scheduling constraints in dial-a-ride problems with transfers : a metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities. *Public Transport*, 9(1-2) :243–272.

[Shaw, 1998] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

[Stephan, 2007] Stephan, H. (2007). Empty seats traveling : Next-generation ridesharing and its potential to mitigate traffic-and emission problems in the 21th century. *Nokia Research Center Bochum*, 14 :2007.

[Toth and Vigo, 2014] Toth, P. and Vigo, D. (2014). *Vehicle routing : problems, methods, and applications*. SIAM.

[Urra et al., 2015] Urra, E., Cubillos, C., and Cabrera-Paniagua, D. (2015). A hyperheuristic for the dial-a-ride problem with time windows. *Mathematical Problems in Engineering*, 2015.

[Vallée et al., 2016a] Vallée, S., Cherif-Khettaf, W. R., and Oulamara, A. (2016a). Algorithme d'optimisation adaptatif pour un service de mobilité intégré. In *ROADEF 2016*.

[Vallée et al., 2016b] Vallée, S., Cherif-Khettaf, W. R., and Oulamara, A. (2016b). Effective strategies for maximizing the number of served requests in a shared dynamic transport system. In *International Symposium on Combinatorial Optimisation*.

[Vallée et al., 2017] Vallée, S., Oulamara, A., and Cherif-Khettaf, W. R. (2017). Maximizing the number of served requests in an online shared transport system by solving a dynamic darp. In *International Conference on Computational Logistics*, pages 64–78. Springer.

[Vallée et al., 2019] Vallée, S., Oulamara, A., and Cherif-Khettaf, W. R. (2019). Reinsertion algorithm based on destroy and repair operators for dynamic dial a ride problems. In *International Conference on Computational Science*. Springer.

[Wong et al., 2014] Wong, K., Han, A., and Yuen, C. (2014). On dynamic demand responsive transport services with degree of dynamism. *Transportmetrica A : Transport Science*, 10(1) :55–73.

[Zhang et al., 2015] Zhang, Z., Liu, M., and Lim, A. (2015). A memetic algorithm for the patient transportation problem. *Omega*, 54 :60–71.