Heuristiques pour la gestion d'une flotte de taxis autonomes, électriques, réservables et partageables

Toussaint Hoché

université
PARIS-SACLAY

Heuristiques pour la gestion d'une flotte de taxis autonomes, électriques, réservables et partageables

Heuristics for the management of a fleet of autonomous, electrics, bookable and shareable taxis

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Unité de recherche : Université Paris-Saclay, UVSQ, Données et Algorithmes pour une Ville Intelligente et Durable, 78035, Versailles, France.
Référent : Université de Versailles-Saint-Quentin-en-Yvelines

Thèse présentée et soutenue à Paris-Saclay, le 9 novembre 2021, par

Toussaint HOCHÉ

Composition du jury

Safia Kedad-Sidhoum Professeure des universités, CNAM	Présidente
Van-Dat Cung Professeur, Université Grenobles-Alpes	Rapporteur et examinateur
Serigne Gueye Maître de recherche, Université d'Avignon	Rapporteur et examinateur
Leila Kloul Maîtresse de conférence, UVSQ	Examinatrice

Direction de la thèse

Dominique Barth Professeur des universités, UVSQ	Directeur de thèse
Thierry Mautor Professeur des universités, UVSQ	Codirecteur de thèse et examinateur
Wilco Burghout Professeur des universités, École royale polytechnique de Suède / Institut Védécom	Coencadrant

Thèse de doctorat

NNT : 2021UPASG082

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Heuristiques pour la gestion d'une flotte de taxis autonomes, électriques, réservables et partageables

Mots clés : Optimisation, simulation, algorithmes, théorie des graphes

Résumé : Le transport se transforme de multiples manières : automatisation des véhicules, électrification, intégration des TIC (Technologies de l'Information et de la Communication) aux systèmes de gestion, ... amenant de nouveaux challenges et opportunités dans le domaine de la mobilité. L'objectif de cette thèse est de mettre au point un service gérant de manière centralisée une flotte de taxis électriques, autonomes, et partagés.

Du point de vue de l'électrification, trois difficultés sont étudiées. 1) La distance pouvant être parcourue après chaque plein est plus faible que pour les véhicules à combustion. 2) un plein est plus long. 3) L'usage de l'infrastructure de recharge est susceptible d'être restreint. La gestion de la recharge des véhicules doit permettre de maximiser le temps de fonctionnement utile des taxis. La gestion de la recharge est intégrée dans un système holistique. On considère ainsi l'impact du stationnement et du partage de course sur le comportement des taxis.

Enfin, nos tests sont effectués sur des instances de grande taille, comptant des dizaines de milliers de clients, basées sur des données réelles. Pour résoudre ces instances, nous proposons une heuristique gloutonne, améliorée par une méthode de recherche de voisinage. Nos résultats montrent la scalabilité de ces méthodes, comment elles doivent être paramétrées, et diverses pistes permettant d'obtenir de meilleurs résultats.

Title : Heuristics for the management of a fleet of autonomous, electrics, bookable and shareable taxis

Keywords : Optimisation, simulation, algorithms, graph theory

Abstract : Transportation is undergoing changes : vehicle automation, electrification, ICT (Information & Communication Technologies) integration in management systems... bringing new challenges and opportunities in the domain of mobility. This thesis objective is to develop a management system for a fleet of electric, autonomous, shared taxis.

Considering electrification, three difficulties are studied. 1) The range of electric vehicles is lower than the one of ICEV. 2) Refilling takes longer. 3) The charging infrastructure may be limited, both in size and in regard to the hours during which it can be used. The management of taxi charging must allow for the maximisisation of taxis' uptime. Charging management is integrated in a holistic system. We thus consider the impact of parking and of ride-sharing on the taxis' behaviour.

Finally, our tests are performed on big instances, with tens of thousands of customers, based on real data. To solve these instances, we propose a greedy heuristic, improved by a neighbourhood search. Our results show the scalability of these methods, how they should be parametered, and various ways to improve results.

Participation

Cette thèse a été effectué dans le cadre d'un partenariat entre le laboratoire DAVID [1] de l'UVSQ [2], membre associé de l'université Paris-Saclay, et l'institut Védécom [3].

Je remercie les membres du jury de thèse, sans qui cette thèse n'aurait pas eu lieu. Je remercie également Tatiana Babicheva et Maxime Redondin, qui m'ont fourni à plusieurs occasions des conseils pertinents vis-à-vis de ma thèse et de mes travaux. Enfin, je remercie toute celles et ceux qui m'ont assisté, de manière plus indirecte ou ponctuelle, durant ces quatres années.

1. Données et Algorithmes pour une Ville Intelligente et Durable
2. Université de Versailles Saint-Quentin
3. institut du Véhicule Décarbonné COMmuniquant

Table des matières

Table des matières

Chapitre 1

La mobilité au XXI^{ème} siècle

Le but de cette thèse est de répondre à des problématiques qui devraient se poser dans le secteur du transport de personnes intra-régional dans les décennies à venir. Ce secteur étant amené à se transformer, il convient de présenter son état actuel, et les grandes tendances qui dirigent son développement. La conclusion de ce chapitre présente le sujet de la thèse.

1.1 TIC et MaaS

Le transport représente une part importante de l'économie. Selon le ministère de la transition écologique et solidaire, il représente 17.3% du PIB français [website, 1]. De ces 17.3%, les trois quarts sont dédiés au transport routier. L'automobile représente plus de 11% des dépenses des ménages.

Le développement des TIC (Technologie de l'Information et de la Communication) est le changement le plus important observé dans ce début de siècle. Ces technologies facilitent grandement la collection d'informations et leur échange, ce qui permet une meilleure estimation des besoins de la population et une meilleure coordination des différents moyens de transports. Un secteur particulièrement affecté par ce changement est celui des Voitures de Transport avec Chauffeur (VTC)[1]. Les VTC utilisaient auparavant principalement des centres d'appel pour connecter clients et chauffeurs, puis des acteurs tels que Uber ou Lyft ont émergé, et avec eux la généralisation des applications smart-phone.

Historiquement, pour avoir la garantie d'obtenir facilement et rapidement un véhicule lorsqu'on le souhaite, la principale solution adoptée a consisté à acheter un véhicule personnel privé. Le développement des TIC incite un nombre grandissant de personnes à passer à un système de "Mobility as a Service" (MaaS). Le principe de MaaS est le suivant : Plutôt que de considérer les moyens de transport comme un *bien de consommation*, considérer l'acte de transport comme un *service*. On observe ainsi actuellement une croissance importante du marché du partage de véhicule : Selon [Preeti and Prasenjit, 2018], ce marché représentait 1.5 milliards de dollars (US) en 2017 et devrait atteindre 12 milliards en 2024. On observe aussi une croissance importante du marché du covoiturage : [website, 5] l'estime à 73 milliards de dollars en 2019, et projette une valeur de 209 milliards en 2025, avec une croissance particulièrement soutenue en Chine et en Inde.

1. anciennement, Voiture de Tourisme avec Chauffeur

Dans la suite de cette thèse, le terme "taxi" est utilisé comme un terme générique pour désigner un véhicule utilisé dans le cadre d'un service de transport de personne. Il peut s'agir d'un VTC, d'un taxi tel que défini par la législation française, d'un véhicule cumulant ces deux activités, ...

1.2 Véhicule autonome

Une autre technologie a le potentiel pour transformer largement et durablement le secteur du transport routier de personnes, l'automatisation des véhicules. Les véhicules autonomes sont aujourd'hui déployés dans des lieux fermés au public (docks, entrepôts, ...). Au cours de cette décennie, et des prochaines, leur usage à de bonnes chances de se généraliser, et d'être déployé à grande échelle dans des lieux publics. La *Society of Automotive Engineers* (SAE) définit 6 niveaux d'autonomie pour les véhicules :

0. **Aucune automatisation** : Le conducteur a un contrôle total et à tout instant des fonctions principales du véhicule (moteur, accélérateur, direction, freins).
1. **Assistance au conducteur** : L'automatisation est présente pour certaines fonctions du véhicule, mais celles-ci ne font qu'assister le conducteur qui garde le contrôle global. Par exemple, le système anti-blocage des roues (ABS) ou l'électro-stabilisateur programmé (ESP) vont automatiquement agir sur le freinage pour aider le conducteur à garder le contrôle du véhicule. Le régulateur de vitesse simple fait également partie de ce premier niveau.
2. **Automatisation de fonctions combinées** : Le contrôle d'au moins deux fonctions principales est combiné dans l'automatisation pour remplacer le conducteur dans certaines situations. Le régulateur de vitesse adaptatif combiné avec le centrage sur la voie fait entrer le véhicule dans cette catégorie, tout comme le Park assist qui permet le stationnement sans que le conducteur n'agisse sur le volant ou les pédales.
3. **Conduite autonome limitée** : Le conducteur peut céder le contrôle complet du véhicule au système automatisé qui sera alors chargé des fonctions critiques de sécurité. Cependant la conduite autonome ne peut avoir lieu que dans certaines conditions environnementales et de trafic (uniquement sur autoroute par exemple). Il est imposé au conducteur d'être en mesure de reprendre le contrôle dans un temps acceptable sur demande du système (notamment lorsque les conditions de circulation autonome ne sont plus réunies : sortie de l'autoroute, bouchon, etc.).
4. **Conduite autonome complète sous conditions** : Le véhicule est conçu pour assurer seul l'ensemble des fonctions critiques de sécurité sur un trajet complet. Le conducteur fournit une destination ou des consignes de navigation mais n'est pas tenu de se rendre disponible pour reprendre le contrôle. Il peut d'ailleurs quitter le poste de conduite et le véhicule est capable de circuler sans occupant à bord. Le véhicule ne peut circuler que dans certaines zones adaptées (geofencing), avec notamment une signalisation adaptées, et éventuellement des équipements de bord de route, comme un réseau 5G transmettant des informations au véhicule.
5. **Conduite équivalente à un humain** : Le véhicule est au moins capable de rouler sur n'importe quelle route publique. Il peut éventuellement être capable de rouler en tout terrain, ou dans des conditions environnementales exceptionnelles (tempête de neige,

route verglacée, travaux avec signalisation contradictoire, ...)

Les niveaux actuellement disponibles pour les consommateurs sont les niveaux 0 à 2. Il est actuellement attendu que les conducteurs aient toujours le contrôle de leur véhicule.

Plusieurs systèmes de taxis autonomes sont actuellement testés dans le monde. General Motors a un service avec une cinquantaine de véhicules à l'essai. Waymo propose à Phoenix (Arizona) un service avec quelques centaines de véhicules autonomes, limités à une zone de quelques kilomètres de côté. Depuis novembre 2019, certains véhicules ne sont plus supervisés par un chauffeur de secours.

Une autonomie entre le niveau 3 et 4 est attendue vers 2024 pour les véhicules haut-de-gamme. Le milieu-de-gamme autonome devra plutôt arriver vers 2030.

Dans cette thèse, nous nous concentrons sur les véhicules de niveau 4 ou plus, qui sont capables de circuler en ville sans chauffeur. Plusieurs études montrent qu'un taxi autonome peut être plus économique que des taxis traditionnels (notamment [Bösch et al., 2018]), sans changements dans les demandes de clients. Si on cumule TIC et autonomisation, on peut donc conclure que d'une part, commander un taxi est bien plus ergonomique qu'avant, et que d'autre part, utiliser un taxi sera moins cher qu'aujourd'hui. [Shoup, 2005] estime que les véhicules personnels passent 95% de leur temps en stationnement. Ce chiffre est relativement stable d'un pays à l'autre. On peut donc conclure que les véhicules personnels sont très peu utilisés. Qui plus est, lorsqu'une voiture est utilisée, le nombre d'occupants à l'intérieur se situe autour de 1.7 dans divers pays. Cette inefficacité a un coût : congestion due à un grand nombre de véhicules transportant peu de personnes, pollution, nombre de m^2 urbains requis pour les places de stationnement important, congestion causée par la recherche de place, ...

1.3 Partage

Cette inefficacité des véhicules personnels a amené des chercheurs à étudier le gain qui pourrait être permis grâce à des systèmes dans lesquels les trajets et les véhicules sont mutualisés. [Burghout et al., 2015] conclut que remplacer les véhicules individuels par des taxis autonomes pourrait permettre à Stockholm une réduction de 92% du nombre de véhicules sans temps d'attente supplémentaire pour les usagers. Cependant, des trajets à vide deviennent nécessaires, causant une augmentation de 24% du nombre de kilomètres parcourus. Mais si le partage de course est permis, en d'autres termes, si plusieurs usagers peuvent utiliser le même taxi simultanément en acceptant des détours, alors une réduction du nombre de véhicules de 95% est possible, avec une réduction du nombre de kilomètres parcourus de 10 à 20%, en fonction de la quantité de détours et de retard toléré. [Martinez et al., 2014] estime qu'une réduction du coût des courses de taxi de 10% peut être obtenue en permettant le partage de course. [Bösch et al., 2018] présente les résultats exposés dans la figure 1.1, et conclut que les véhicules privés sont actuellement moins chers que les taxis car un chauffeur est nécessaire, mais que des taxis autonomes, même sans partage de course, sont moins chers que des véhicules privés. L'auteur note toutefois qu'un marché important peut persister pour les véhicules privés, même s'il est difficile de l'évaluer, car les régulations, infrastructures et usages sont susceptibles de changer largement. L'auteur note

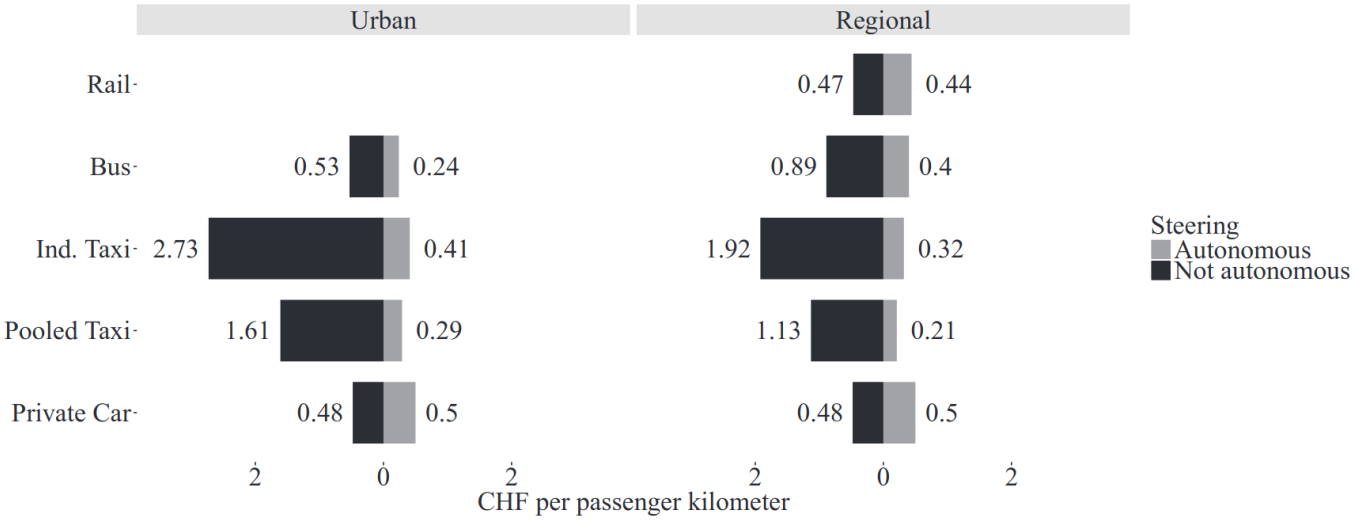


Figure 1.1 – *extraite de [Bösch et al., 2018]. Comparaison des coûts en franc suisse par kilomètre de différents modes de transport, en ville et hors-ville. Sont considérés : Train, bus, taxi individuel, taxi avec partage de course, et voiture privée, en fonction de si le véhicule est autonome ou non.*

également qu'avec l'automatisation, les lignes de bus sont susceptibles de changer, avec une augmentation de leur fréquence mais une diminution de la taille des véhicules.

1.4 Temps morts et Stationnement

Deux changements supplémentaires sont envisageables. Actuellement, les courses nocturnes sont plus coûteuses. La raison est triple : 1) Le travail de nuit est désagréable/contraignant pour les chauffeurs, qui requièrent donc une compensation. 2) Le nombre de clients est plus faible, donc les chauffeurs doivent attendre plus longtemps entre chaque course, et doivent rouler plus longtemps à vide avant et après chaque course. 3) Comme le nombre de clients est faible, le nombre de taxis est lui aussi faible, donc le taxi le plus proche d'un client peut être loin. Dans le cas des taxis autonomes, La première et la troisième barrières sautent. De fait, un propriétaire de flotte de taxis autonomes à peu intérêt à ce que ses taxis soient inactifs, leur coût opérationnel étant faible. On peut donc supposer que les taxis autonomes sont un moyen de transport particulièrement intéressant la nuit, car beaucoup de taxis sont disponibles et peu chers avec peu de congestion. Cela signifie également que, aux heures où il n'y a pas beaucoup de personnes à transporter, un taxi peut générer un profit en transportant des marchandises. Plusieurs programmes ont été menés dans différentes villes pour développer les livraisons nocturnes [Ukkusuri et al., 2015, Taefi, 2016]. Leur succès a été mitigé, mais pour les entreprises ayant mis en place un système automatisé de réception de colis, les résultats se sont montrés encourageants et durables dans le temps.

La demande en taxi est très variable au cours d'une journée. Le transport de biens, en revanche, est généralement plus flexible du point de vue de l'heure de livraison, et est donc complémentaire au transport de personne : il peut lisser le taux d'utilisation des taxis au cours de la journée.

Concernant le stationnement, il peut être intéressant d'augmenter le nombre de taxis

dans les villes où il s'agit d'un problème critique. En effet, [Shoup, 2005] montre que, sur 16 études menées en centre-ville entre 1927 et 2001, en moyenne 30% des véhicules sont à la recherche d'une place de stationnement. Bien que ces chiffres soient à la fois anciens et discutables, ils montrent que la recherche d'une place est un facteur significatif pour expliquer la congestion urbaine. Puisque les taxis passent moins de temps stationnés que les voitures privées, réduire le nombre de places de stationnement serait possible, et pourrait réduire la congestion (sans compter que retirer des places de stationnement peut permettre l'ajout de nouvelles voies de circulation). Cela étant, diverses études ont montré que l'augmentation des VTC a plutôt causé une augmentation de la congestion, car ils concurrencent les transports en commun.

En supposant donc que les taxis autonomes et partagés vont se développer à l'avenir, on peut alors se demander comment ils seront gérés. Depuis leur apparition, les chauffeurs de taxi utilisent une approche individualiste : chaque chauffeur cherche à maximiser ses propres gains en faisant en sorte d'être toujours dans des zones où le nombre de clients est élevé et la concurrence faible, tout en minimisant les trajets à vide. Un tel comportement peut être transposé pour des taxis autonomes, il s'agirait alors d'un modèle multi-agent. Mais les raisons pour lesquelles les taxis fonctionnent ainsi sont que d'une part la communication entre taxis est limitée, et donc que chaque chauffeur doit agir de manière largement indépendante, en utilisant son expérience et sa connaissance du terrain, et d'autre part que les chauffeurs sont en concurrence, donc la coopération est limitée.

On peut donc se demander si un système multi-agent est réellement le meilleur qui puisse être implémenté. Pour optimiser les bénéfices de l'ensemble de la flotte, et puisque les TIC permettent une communication virtuellement instantanée et permanente, un système centralisé contrôlant chaque action de chaque taxi peut être préférable, si la législation et le temps de calcul le permettent.

1.5 Électrification

Une dernière évolution majeure dans le domaine du transport routier est l'électrification des véhicules. Plusieurs pays, dont la France et le Royaume-Uni ont mis en place un calendrier pour l'arrêt de la vente de véhicules consommant des énergies fossiles [website, 2]. Bien que seuls 3% des véhicules vendus en 2019 soient électriques, cette proportion croît à un rythme élevé (fois six entre 2014 et 2018 mondialement). Un certain nombre de villes prennent également des mesures pour limiter l'accès au centre-ville aux véhicules diesel et essence, visant à terme un bannissement de ces véhicules. Au niveau national, les raisons annoncées sont la réduction de gaz à effet de serre et le respect de l'accord de Paris. Au niveau des villes, l'amélioration de la qualité de l'air est le principal argument avancé. Parmi les alternatives au véhicule essence, le véhicule électrique est la solution la plus étudiée.

En termes d'usage, un véhicule électrique est actuellement très différent d'un véhicule essence. Il y a deux raisons à cela.

Limite de portée

Premièrement, les voitures électriques ont une portée limitée. Prenons par exemple la voiture électrique la plus vendue au monde : la compacte 5 portes Nissan Leaf. Selon

l'USEPA [2], la version la plus récente de cette voiture (2016) a une consommation de 17.4kWh pour 100km, pour une portée de 172km avec la batterie la plus grande (30kWh), ce qui est insuffisant pour un aller-retour Paris-Chartres. De plus, il est connu que les véhicules électriques ont une portée plus limitée en hiver, car 1) le chauffage de l'habitacle consomme de la batterie, et 2) les batteries lithium-ion utilisées dans les véhicules électriques fonctionnent mal par grand froid. [Delos Reyes et al., 2016] a conclu suite à des expérimentations que pour maintenir une température de 21°C dans l'habitacle avec une température extérieure de -15°C, la portée doit grossièrement être divisée par deux. Une étude de 2020 de la NAF [3] indique qu'une diminution de la portée d'environ 20% est observée entre 0°C et -5°C par les 20 véhicules les plus vendus en Norvège.

Qui plus est, les batteries s'usent avec le temps. Les constructeurs automobiles proposent une location de batterie, et garantissent leur remplacement lorsque la charge maximale diminue en dessous d'un certain seuil, aux alentours de 75%. Par une estimation grossière, on peut donc conclure qu'une voiture électrique avec une batterie usée a, par -15°C, une autonomie réduite de 63% par rapport à sa portée annoncée. La Leaf 2016 30kWh est ainsi capable dans ces conditions de parcourir 65 kilomètres, soit une portée insuffisante pour un aller-retour Roissy-Le Louvre. La voiture électrique avec la plus grande portée en 2020 est la Tesla modèle S, version 100kWh. avec une portée d'environ 600km (selon l'USEPA), il peut être impossible avec elle d'effectuer un aller-simple Paris-Le Mans. On peut dès lors supposer que, pour compenser ce manque de portée, il est nécessaire de pouvoir se recharger rapidement et partout. Ce qui nous amène à la seconde différence majeure entre véhicule essence et véhicule électrique.

Recharge

En oubliant un instant le fait qu'il n'y a, à ce jour, pas d'interface chargeur / véhicule autonome [4], le principal problème concernant la recharge est que la vitesse de réapprovisionnement des véhicules électriques est beaucoup plus lente que pour les véhicules essence. Une pompe à essence délivre environ 36 litres d'essence par minute. Une voiture essence 5 portes consomme environ 6L/100km. Autrement dit, un véhicule essence regagne 600km d'autonomie chaque minute passée à la pompe. Pour les véhicules électriques, les chargeurs commerciaux les plus rapides ont actuellement une puissance de l'ordre de 120kW (soit 2 kWh/min). Les voitures électriques 5 portes ont une consommation supérieure à 15kWh/100km. Il faut donc au moins 7.5 minutes pour donner 100 km de portée, ce qui est 45 fois plus lent.

Pour compenser ce problème, des chargeurs plus rapides pourraient être déployés. Pour qu'un chargeur électrique soit aussi rapide qu'une pompe à essence, il faudrait qu'il ait une puissance d'au moins 5.4MW. A titre de comparaison, les réacteurs nucléaires opérationnels en France ont une puissance moyenne de l'ordre de 1000 MW ; Autrement dit, 1 réacteur nucléaire $\approx$ 200 "pompes à essence électriques". On peut alors se demander si une massification des véhicules électrique est seulement possible.

D'après le SDES [5], les français ont réalisé 756 milliards de kilomètres-voyageurs en 2016

2. United States Environmental Protection Agency
3. Norwegian Automobile Federation
4. A l'exception de la recharge sans fil
5. Service des Données et Études Statistiques

en transport individuel. Si on suppose qu'une voiture contient en moyenne 1.5 voyageur, et une consommation de 15 kWh pour 100 km, cela représente 75 milliards de kWh, soit 75 TWh. En comparaison, la France avait une production nette d'électricité de 529 TWh en 2017, selon le ministère de la transition énergétique et solidaire. Si on ajoute les 267 milliards de tonnes-kilomètres de transport routier de marchandises, et les 200 milliards de kilomètres-voyageurs des transports routiers collectifs, on constate qu'une croissance significative du réseau électrique est nécessaire, tant au niveau de la production que de l'infrastructure de distribution, pour remplacer les véhicules actuels par des véhicules électriques.

Il paraît donc judicieux de supposer que la recharge des véhicules électriques sera sensiblement plus restreinte que le réapprovisionnement des véhicules à combustion (en interdisant par exemple la recharge en heure pleine), et aussi sans doute que des changements majeurs vont avoir lieu dans le domaine de la mobilité, avec une utilisation plus faible de la voiture, au profit du distanciel et des transports en commun.

Ce problème peut être particulièrement gênant dans les pays en transition vers une production électrique intégralement renouvelable, ce qui occasionne déjà des problèmes, comme l'irrégularité et l'imprévisibilité de la production éolienne et solaire, le remplacement des centrales à charbon et la réorganisation/décentralisation massive de la production électrique. Cela étant, la France se base principalement sur le nucléaire pour sa production électrique (75% de la production, contre 20% dans le reste du G20), et qu'il s'agit d'une source d'énergie à la fois longue et coûteuse à abandonner, en plus d'être faible émettrice de CO_2. La France devrait donc être moins affectée dans un futur proche. Le calendrier actuel de l'État français présente cependant une réduction de la part du nucléaire à 50% en 2035.

Certaines techniques, comme l'échange de batterie ou la charge par induction en mouvement, pourraient permettre de réduire le besoin de chargeurs ultra-rapide tout en lissant la recharge de batteries dans le temps, mais nous verrons dans le chapitre 2 qu'elles ne sont pas adaptées à tous les cas de figure.

Malgré ces points négatifs, et comme il a été mentionné, il est attendu que les parts de marché du véhicule électrique augmentent massivement au cours des deux prochaines décennies, avec notamment la fin de la vente de véhicule essence en Union Européenne en 2035. Il est également attendu que des progrès importants aient lieu concernant les limites actuelles des véhicules électriques, notamment leur portée, même s'il est encore incertain qu'ils atteignent les capacités des véhicules essence. Il est donc important de mettre au point des stratégies pour réduire l'impact de ces limitations, et d'établir quels usages sont envisageables pour ces véhicules.

1.6 Conclusion

Résumons maintenant les caractéristiques principales des systèmes de transports des prochaines décennies, et en particulier la place des véhicules autonomes dans ces systèmes.

1.6.1 La mobilité au XXI^{ème} siècle

Concernant les trajets de courte distance (< 100 km) qui auront lieu au milieu du siècle, voici les prédictions les moins contestées, en supposant une généralisation des véhicules autonomes.

Trois grandes figures se dégagent : l'usage rural, péri-urbain et urbain (*i.e.* en zone à faible, moyenne, et forte densité de population).

La densité de population a un impact majeur sur l'efficacité économique d'un type donné de transport. De manière générale, plus la densité est élevée, plus les transports de masse se justifient.

Supposons par exemple qu'on a une zone de 10×10km. Dans cette zone, 10 personnes souhaitent se déplacer avec un unique véhicule partagé (taxi, bus, ...). Ce véhicule devra se déplacer dans toute la zone pour les récupérer [6]. Si le nombre de véhicules et de voyageurs est multiplié par 100, alors cette zone peut être découpée en 100 sous-zones de 1×1km. Chaque véhicule a toujours dix personnes à récupérer, mais dans une zone 100 fois plus petite, ce qui implique une diminution du temps d'attente des voyageurs et du carburant utilisé par le véhicule.

En zone rurale (faible densité), la voiture privée devrait rester compétitive, avec peu d'options de transports en commun [website, 9, Bösch et al., 2018].

Dans les zones à forte densité, l'efficacité des transports de masse (train, bus, métro, tram, ...) est susceptible d'augmenter sensiblement avec le développement des TIC [Bösch et al., 2018, website, 14], permettant le développement de systèmes de transport multimodaux [website, 14].

Ce développement technologique est également susceptible de rendre compétitifs certains de ces modes de transports dans des zones à moyenne densité. Un point en particulier qui favorise le développement des transports de masse est le développement des transports en libre-service, comme Velib ou Autolib, qui complémentent bien les transports de masse et leur rigidité spatio-temporelle.

La situation des zones à moyenne densité est plus nuancée, et les solutions proposées seront sans doute à étudier plus au cas-par-cas.

1.6.2 Usage des véhicules autonomes

Quel sera donc l'usage des véhicules autonomes dans ce contexte ?

Il faut une densité critique pour qu'un système de taxis partagés soit compétitif face à des véhicules privés individuels. Un service de véhicules autonomes peut être envisagé pour une communauté (*e.g.* membres d'une même résidence), mais le véhicule individuel privé devrait rester majoritaire. Si ce véhicule est autonome, se sera vraisemblablement avant tout pour une question de confort du propriétaire.

Dans les zones à forte densité, les véhicules autonomes peuvent être à la fois complémentaires des transports de masse, et en compétition avec eux. Pour améliorer efficacement la mobilité des habitants, il paraît important d'intégrer les véhicules autonomes aux autres systèmes de transport, plutôt que de mettre en place un réseau parallèle. Il est cependant incertain qu'un tel réseau devienne la norme. En particulier, un acteur comme Uber n'a pas nécessairement intérêt à lier intimement son réseau privé au réseau public.

Dans les zones à moyenne densité, un système indépendant de taxis autonomes partagés peut être pertinent. Qui plus est, des revenus complémentaires, par exemple avec un transport de biens (comme présentés dans la section 1.4) pourraient permettre dans certaines villes à une flotte de taxis d'atteindre une masse critique. Il est donc possible que dans

6. en supposant une distribution spatiale équilibrée des clients

certaines villes de moyenne densité, les taxis autonomes deviennent le principal mode de transport.

1.6.3 Le sujet de cette thèse

L'utilité des taxis est fortement dépendante de la densité de population. Cette thèse étudie en particulier le cas des zones de moyenne densité, qui sont à la fois suffisamment denses pour qu'un service de taxis partagés soit viable économiquement (hors marché de niche), et en même temps trop peu denses pour qu'il y ait une véritable influence des transports de masse.

En résumé, l'objectif de cette thèse est de mettre au point un service gérant exclusivement une flotte de taxis électriques et partagés. Ces taxis sont autonomes, et donc, entre autres choses, compatibles avec l'usage d'un système centralisé. Cette flotte compte un nombre de véhicules de l'ordre de quelques dizaines ou de quelques centaines.

L'infrastructure de recharge est restreinte, notamment, les chargeurs sont partagés à la fois entre les taxis et avec d'autres véhicules particuliers. Le but premier de cette thèse est de faire en sorte que la nécessaire recharge des taxis perturbe le moins possible le service.

Chapitre 2

Positionnement du problème

Suivant le contexte établi dans le chapitre précédent sur les évolutions du transport de personnes, cette thèse a pour objet de proposer des méthodes de gestion d'une flotte de taxis autonomes et électriques pour laquelle le partage de course est autorisé. Il conviendra pour cela de gérer complètement l'activité de chaque taxi, *i.e.* non seulement lui attribuer les courses qu'il doit accomplir, mais également prévoir les trajets à vide, le stationnement et la recherche de place de stationnement, mais surtout les recharges de batterie qu'il doit effectuer.

Les problèmes liés à la gestion d'une flotte de véhicules ont été largement étudiés depuis longtemps. Le problème étudié dans cette thèse partage des similitudes avec certains de ces problèmes, mais est original sous certains aspects.

Dans ce chapitre, les principaux problèmes de gestion de flotte de véhicules sont présentés, et le problème étudié dans cette thèse est positionné, *i.e.* sa relation avec ces autres problèmes est explicitée.

2.1 Principaux problèmes de transports

2.1.1 Problème de voyageur de commerce (TSP)

Un problème à la racine de nombreux problèmes de gestion de véhicules est le TSP (Travelling Salesman Problem). Ce problème a pour origine la question suivante : "Ayant une liste de villes et connaissant la distance entre chaque paire de villes, quel chemin un voyageur doit-il emprunter pour visiter chaque ville et terminer à son point de départ, en minimisant la distance qu'il parcourt (voir figure 2.1)?" Le TSP peut être modélisé comme un problème de graphe ou un programme linéaire. Voici sa modélisation en problème de graphe.

Entrée Un graphe $G = (V, A)$ pondéré et complet, dans lequel chaque sommet représente une ville et chaque arc une route entre deux villes.

Sortie Un circuit élémentaire c dans G, passant par tous les sommets, qui représente la tournée du voyageur de commerce.

Objectif L'ensemble des arcs dans c doit avoir le plus petit poids total (*i.e.* minimiser la distance parcourue durant la tournée).

Figure 2.1 – *Un TSP et une tournée passant par 15 villes allemandes. Image extraite de Wikipédia.*

Dans sa formulation originelle, le graphe G n'est pas nécessairement complet. Dans ce cas, il est cependant possible d'ajouter des arcs de poids infini à G pour qu'il devienne complet, ce qui ne change pas la solution si G contenait un circuit hamiltonien de poids fini avant modification.

Le TSP est un problème NP-difficile dont la version en problème de décision est NP-complète au sens fort. Dans la suite de ce chapitre, il existe une réduction polynomiale du TSP vers tous les autres problèmes mentionnés. En d'autres termes, tous les problèmes de ce chapitre sont aussi au moins NP-difficiles et leur version en problème de décision est au moins NP-complète au sens fort.

Résolution du TSP

De nombreuses méthodes ont été développées pour résoudre ce problème. Les approches les plus classiques sont recensées dans [Lawler et al., 1991]. Certaines sont dédiées spécifiquement au TSP, comme l'heuristique 2-opt [Croes, 1958], d'autres sont des méthodes d'optimisations combinatoires diverses. Le solveur Concorde de l'université de Waterloo est généralement considéré comme le plus efficace, utilisant une méthode de branch-&-cut [David et al., 2006] à la fois pour le TSP et pour certaines de ces variantes. Les articles les plus récents se concentrent sur les heuristiques, en particulier celles du type intelligence distribuée [Raman and Gill, 2017] comme la PSO [1] ou sur des algorithmes offrant une garantie sur la qualité de l'approximation [Karpinski et al., 2015].

1. PSO : Particle Swarm Optimisation / Optimisation par essaims particulaires.

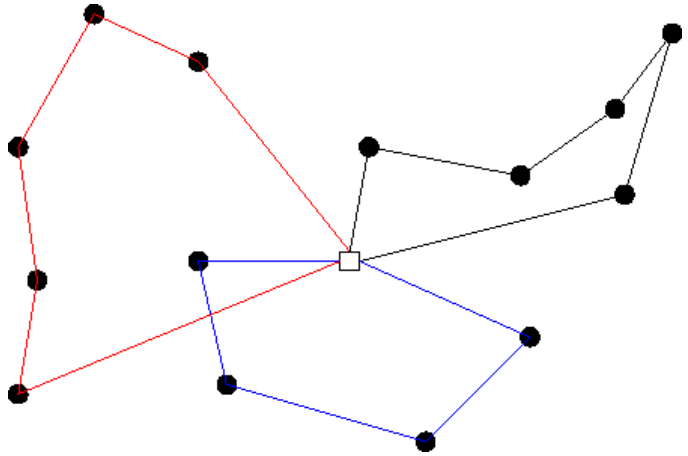


Figure 2.2 – *Un VRP avec 3 véhicules dans un graphe avec 14 sommets. Image extraite de Wikipédia.*

2.1.2 Problèmes de tournées de véhicules (VRP)

Une extension naturelle au TSP et de considérer qu'on ne dispose plus d'un seul véhicule mais d'une flotte de véhicules. On parle alors de problème de tournée de véhicules (VRP : Vehicle Routing Problem). La plupart des applications concrètes de ce problème consistant en la livraison d'un ensemble de clients, des contraintes additionnelles sont à prendre en compte. Les deux plus classiques sont celles liées à la capacité des véhicules et à la disponibilité des clients. Cette dernière est généralement exprimée sous la forme d'une fenêtre de temps (Time Window) durant laquelle un véhicule peut arriver chez le client pour le livrer. Cela donne donc une version du problème qu'on appelle CVRPTW (Capacitated VRP with Time Windows). De très nombreuses variantes existent tant au niveau des contraintes que de l'objectif (on peut par exemple vouloir minimiser le nombre de véhicules utilisés). Au final, la littérature sur le VRP est prolifique, comme le montrent les revues de [Braekers et al., 2016, Eksioglu et al., 2009, Pillac et al., 2013].

Le cœur du VRP peut être défini ainsi :

Entrée Un graphe pondéré $G = (V, A)$, dans lequel chaque sommet représente un client et chaque arc une route. Un sommet $v \in V$ qu'on appelle le dépôt. Un nombre de véhicules k dans ce dépôt.

Sortie Un ensemble C de k circuits dans G, représentant la tournée de chaque véhicule. Chaque sommet de G est dans au moins un circuit de C, et tous les circuits passent par le dépôt.

Objectif L'ensemble des arcs dans C doit avoir le plus petit poids total (*i.e.* minimiser la distance totale parcourue par les véhicules).

Résolution du VRP

Les méthodes utilisées pour le TSP sont pertinentes pour certaines variantes du VRP [Fukasawa et al., 2006]. Cela étant, les heuristiques spécifiquement dédiées au TSP perdent en pertinence au fur et à mesure que le VRP est étendu. Une approche populaire avec le VRP mais peu avec le TSP est la génération de colonnes, proposée en 1964 et largement réutilisée depuis [Simchi-Levi et al., 2014, Behnke et al., 2021].

Une revue de la littérature de [Braekers et al., 2016] indique que plus des deux-tiers des articles parus sur le VRP et ses variantes sont consacrées à des méta-heuristiques. Parmi les approches populaires de ces dernières années, on retrouve la PSO [Marinakis et al., 2018] et le Machine-Learning [Bai et al., 2021].

2.1.3 Pick-up & Delivery Problem (PDP)

Le Pick-up & Delivery Problem (PDP) est une variante de VRP, dans laquelle l'objectif n'est pas de passer par un ensemble de points de passages, mais récupérer un ensemble d'objets puis les déposer ailleurs. On peut représenter une instance de ce problème à l'aide d'un graphe. Dans ce graphe, chaque sommet est associé à un stock (*e.g.* contient 2 tonnes de pommes et 1 tonne de radis) et chaque sommet est aussi associé à une commande (*e.g.* A besoin de 0.5 tonnes de radis). Pour chaque type d'objet, le stock total est égal à la demande totale. Un sommet de ce graphe est le dépôt, qui contient des véhicules. Le but est de générer un circuit pour chaque véhicule de sorte que la commande de chaque sommet soit livrée.

Cette description correspond à la forme générale du PDP, qu'on appelle aussi *many-to-many* PDP. Dans ce mémoire, on s'intéresse plus particulièrement à la variante *one-to-one*, dans laquelle chaque objet a une origine et une destination précise. Le PDP n'a pas de définition exacte, par exemple, la définition donnée dans [Savelsbergh and Sol, 1995] est différente de celle donnée dans [Berbeglia et al., 2007]. Voici une définition du one-to-one PDP dans sa forme la plus basique.

Entrée On a un graphe pondéré $G = (V, A)$. L'un des sommets de V est le dépôt. On a k véhicules. On a un ensemble d'objets. Chaque sommet de V autre que le dépôt est soit l'origine soit la destination d'un objet précis. Chaque objet a une unique origine et une unique destination.

Sortie Un ensemble C de k circuits dans G couvrant tous les sommets. Chaque circuit représente la tournée d'un véhicule. Si un véhicule passe par l'origine d'un objet, alors il doit aussi passer par la destination de cet objet plus tard durant sa tournée.

Objectif Minimiser le poids total des arcs de C (la distance totale parcourue par les véhicules).

Dans la suite de ce mémoire, le terme PDP désigne le one-to-one PDP tel qu'il vient d'être décrit. La figure 2.3 présente un exemple de PDP. Comme pour le VRP, de nombreuses variantes du PDP existent, dont les principales sont présentées par [Berbeglia et al., 2007, Berbeglia et al., 2010]. Une variante dédiée au transport de personnes est le Dial-A-Ride Problem (DARP).

Résolution du PDP

Dans sa définition la plus simple, les méthodes utilisées pour résoudre le PDP sont relativement similaires à celles utilisées pour le VRP. Pour les variantes du PDP, les méthodes utilisées sont largement dépendantes de la variante considérée. Présentons le DARP, qui est une des extensions les plus courantes du PDP.

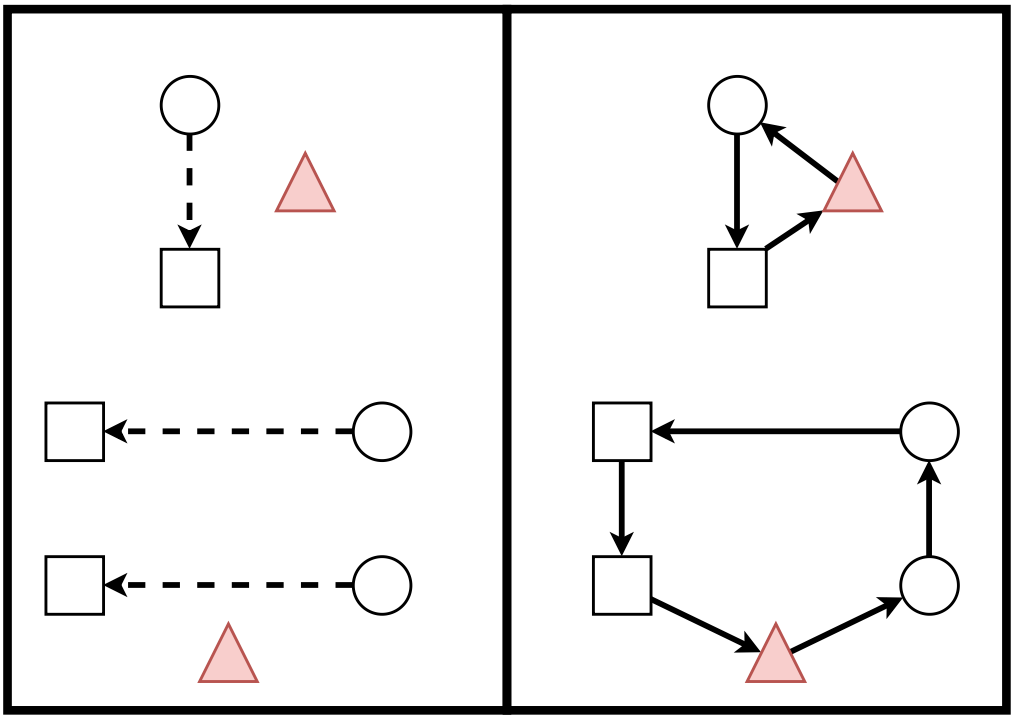


Figure 2.3 – *A gauche, une instance du PDP avec deux véhicules (triangles) et trois objets répartis dans un plan. Un rond représente l'origine d'un objet, et le rectangle auquel il est lié par un trait pointillé et la destination de cet objet. A droite, une solution optimale pour cette instance du PDP.*

2.1.4 Dial-A-Ride Problem (DARP)

Les services de transport à la demande (DRT / Demand-Responsive Transports), ou Dial-A-Ride en anglais, sont un mode de transport en commun à itinéraire variable. Ce mode de transport est à mi-chemin entre la ligne de bus et le taxi individuel. Chaque usager indique à l'avance le trajet qu'il souhaiterait effectuer, ainsi que son heure de départ. Le gestionnaire du service génère ensuite un parcours pour chaque véhicule de sa flotte de sorte que chaque usager soit récupéré et déposé en respectant un ensemble de contraintes de qualité de service.

Les trajets empruntés par les véhicules ne sont pas prédéterminés. Les véhicules utilisés sont de moyenne capacité (mini-bus ou vans) pour permettre un usage partagé simultané. Il s'agit généralement d'un service public plutôt qu'un service commercial, et il a souvent pour utilisateurs des personnes à mobilité réduite.

Ce type de service peut naturellement être modélisé sous la forme d'un PDP, dans lequel chaque objet est remplacé par un usager. On appelle Dial-A-Ride Problem (DARP) un PDP spécialisé sur le transport de personnes. Deux modifications doivent être apportées au PDP pour en faire un DARP. La définition de DARP choisie est celle présentée par [Cordeau and Laporte, 2003].

Ajouts pour passer du PDP au DARP

Premièrement, les transports de personnes ont un nombre limité de places. Deuxièmement, la récupération d'un usager et sa dépose doivent se faire dans une fenêtre de temps.

Par exemple, un usager souhaitant partir à 9h00 ne peut pas être récupéré à 7h00, et il ne peut pas non plus être déposé à 19h00 alors qu'il pouvait raisonnablement espérer que son trajet ne dure que 20 minutes. Dans le DARP, chaque usager a donc une fenêtre de temps durant laquelle il peut être récupéré et déposé.

Un PDP avec contrainte de fenêtre de temps est appelé un PDPTW (PDP with Time Windows). Il n'est pas garanti qu'une solution traitant tous les usagers existe. Par exemple, il est possible que $k+1$ usagers doivent être récupérés en même temps à des endroits différents, ce qui est impossible avec k véhicules [2]. Pour cette raison, l'objectif doit être changé.

Il existe plusieurs variantes du PDPTW, avec différents objectifs. En règle générale, la valeur d'une solution baisse pour chaque usager non satisfait. Il est courant de faire rentrer le retard de chaque usager dans la fonction objectif, et dans ces cas, l'heure d'arrivée n'est pas forcément une contrainte stricte, dans la mesure où un usager déposé très tard provoque simplement une baisse de la valeur de la solution. [Cordeau and Laporte, 2007] propose une revue de la littérature sur le DARP.

On peut définir un DARP comme un CPDPTW. La distinction entre ces deux problèmes vient principalement de leurs variantes. Les variantes du DARP sont plus axées sur le transport de personnes, par exemple, les usagers doivent parfois être déposés dans l'ordre dans lequel ils sont récupérés, ou bien un usager n'accepte qu'un nombre limité d'usagers avec lesquels il partage sa course, ou d'arrêts durant sa course.

Détails sur la modélisation des requêtes

Beaucoup d'articles sont apparus ces dernières années concernant le DARP. Cela est notamment dû au développement des modes de transports facilités par les smart-phones. [Ho et al., 2018] présente les travaux récents sur le DARP.

Lorsque ce n'est pas précisé, un DARP est statique, *i.e.* les données d'entrées du problème sont statiques, sans inconnues. Les problèmes statiques (*offline*) s'opposent aux problèmes (*dynamiques*) dans lesquels les entrées du problème sont découvertes progressivement. Par exemple, dans le cas d'un DARP dynamique, les usagers ont un instant de formulation, et le gestionnaire de flotte de véhicules ne découvre la demande d'un usager qu'à cet instant. Gérer une flotte de taxis traditionnels revient fondamentalement à résoudre un VRPTW dynamique dans lequel chaque sommet représente une course, et un arc représente le trajet à vide entre la destination d'une course et l'origine de la suivante.

La figure 2.4 présente une requête d'un usager et son traitement dans un DARP dynamique.

Une requête est d'abord formulée, *i.e.* découverte par le gestionnaire de la flotte qui ignorait son existence jusque-là. Le gestionnaire de la flotte doit ensuite décider s'il accepte ou non de traiter la requête. On distingue à ce stade trois variantes du DARP dynamique.

1. Le gestionnaire doit donner une réponse définitive à chaque usager en temps réel. Dans ce cas, le gestionnaire essaye d'insérer une-par-une chaque requête dans les emplois du temps des véhicules.
2. Le gestionnaire peut attendre un peu avant de donner une réponse. Dans ce cas, les requêtes peuvent être regroupées, et les requêtes formulées à peu près en même temps sont insérées simultanément dans les emplois du temps des véhicules.

2. Ce constat est aussi vrai pour le VRPTW.

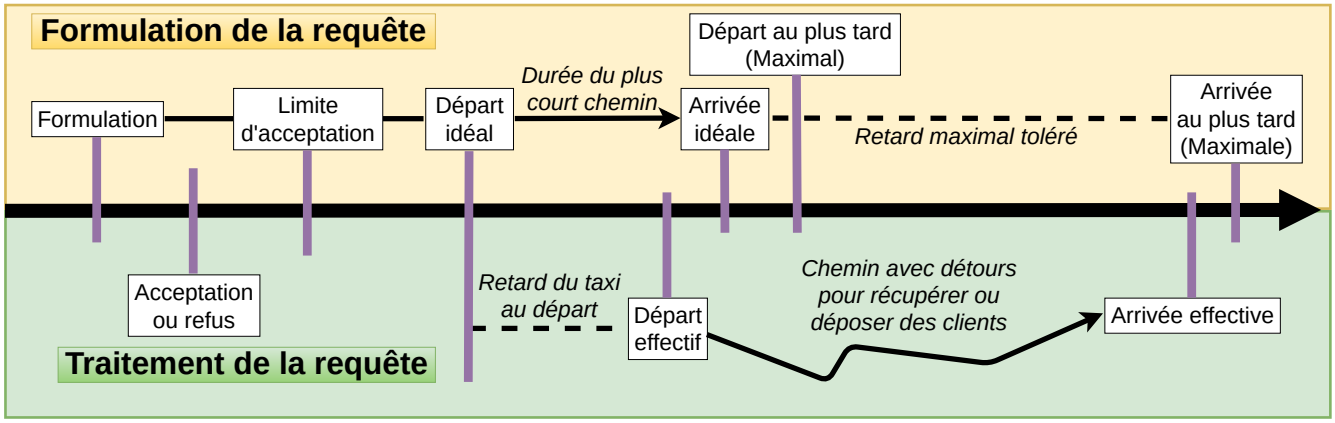


Figure 2.4 – *Frise chronologique d'une requête et de son traitement dans un DARP. L'heure de formulation n'existe que dans le problème dynamique (temps réel).*

3. Le gestionnaire n'a jamais à indiquer à un usager s'il compte le traiter. Dans ce cas, les usagers ne réservent généralement pas, et souhaitent être récupérés le plus tôt possible (l'heure idéale de départ est l'heure de formulation).

Chaque usager indique une origine et une destination, et, généralement, l'heure à laquelle il souhaite partir. Connaissant cette heure, le gestionnaire peut indiquer à quelle heure l'usager peut raisonnablement espérer arriver.

L'ensemble des indicateurs temporels de la figure 2.4 peuvent être déterminés à partir de ces informations, ou bien faire partie intégrante de la requête d'un usager. Par exemple, l'heure maximale de départ peut être :

— donnée par l'usager, ou cachée par l'usager. Dans ce second cas, le gestionnaire doit proposer une heure de départ, et l'usager peut annuler sa requête si l'heure de départ est trop tardive.

— déterminée à partir de l'heure idéale de départ et d'une garantie de qualité de service du gestionnaire. Par exemple, tout usager est récupéré dans une fenêtre de 5 minutes suivant son heure idéale de départ. L'heure maximale d'arrivée est alors généralement limitée par un nombre ou une durée maximum des détours durant la course.

— déterminée à partir de l'heure maximale d'arrivée, *i.e.* toute heure de départ est bonne du moment que l'usager arrive avant son heure maximale d'arrivée.

L'heure maximale d'arrivée n'est pas présente dans tous les modèles, principalement pour deux raisons. Premièrement, cette modélisation permet de garantir l'existence d'une solution traitant tous les usagers. Dans ce cas, l'objectif est de maximiser une fonction qui utilise le retard comme une pénalité. Deuxièmement, cette modélisation permet de représenter une gestion réelle de flotte de véhicules partagés. En effet, il est rare que les gestionnaires de telles flottes offrent une véritable garantie sur l'heure d'arrivée, notamment à cause de l'imprévisibilité du trafic. Dans ce cas, les détours sont limités par une distance et/ou un nombre d'arrêts, plutôt qu'une durée.

Dans une variante moins commune du DARP, les usagers indiquent l'heure à laquelle ils souhaitent arriver plutôt que l'heure à laquelle ils souhaitent partir, et le gestionnaire calcule à quelle heure chaque usager peut être récupéré au plus tôt.

Variante du DARP la plus proche du problème étudié dans cette thèse

Voici la définition du DARP utilisée dans ce mémoire.

Entrée Un graphe complet $G = (V, A)$ dans lequel chaque arc a une durée. Un des sommets est appelé le dépôt. Un ensemble d'usagers, tel que chaque sommet de V est l'origine ou la destination d'exactement un usager, ou est le dépôt. Une fenêtre de temps est associée à chaque sommet sauf le dépôt. Un ensemble de k véhicules de capacité c.

Sortie Un ensemble de k circuits dans G passant par le dépôt, tel que chaque circuit représente la séquence de récupérations et de déposes de passagers d'un véhicule. Le nombre de passagers dans un véhicule ne doit jamais dépasser c. Si un véhicule passe par l'origine d'un usager, alors il doit aussi passer par la destination de cet usager par la suite. Les véhicules peuvent attendre dans chaque sommet, car un véhicule ne peut quitter un sommet que durant la fenêtre de temps associée à ce sommet.

Objectif Maximiser le nombre d'usagers traités.

Une revue de la littérature sur le DARP dynamique est présentée par [Agatz et al., 2012], et une revue sur le partage de véhicules est présentée par [Jorge and Homem de Almeida Correia, 2013], dont des articles sur le partage de course. Généralement, dans les articles mentionnés, les clients commandent un taxi au dernier moment, plutôt que de réserver à l'avance.

Trouver des articles pratiques traitant du partage de course est ardu, principalement parce que le terme *partage de course* n'a pas d'équivalent en anglais. Le partage de course, en français, est l'acte de partager un taxi (ou tout autre véhicule commercial à faible capacité) pour la durée d'une course [3]. Jusque dans les années 2000, le terme *ride-sharing* était un synonyme exact de *partage de course*, mais le sens de ce mot a changé avec l'arrivée de compagnie comme Uber. Le ride-sharing peut maintenant aussi désigner un système dans lequel des particuliers travaillent comme taxis indépendants, et dans la plupart des pays sans avoir le statut officiel de chauffeur de taxi. La littérature récente sur le partage de course est donc moins fournie qu'il ne pourrait sembler de prime abord.

Types d'instances

Pour le TSP, le VRP, le PDP et le DARP, les algorithmes et heuristiques développées sont principalement testées sur des instances basées sur le banc d'essai de [Solomon, 1987]. dans ces instances, il y a quelques dizaines voire quelques centaines de points de passage ou requêtes. Ces instances sont également complètement artificielles. La revue de la littérature sur le VRP et ses extensions de [Braekers et al., 2016] conclut qu'environ 5% des articles d'avant 2016 sont sur des instances artificielle (du type [Solomon, 1987]).

La qualité et quantité des données disponibles depuis quelques années a ouvert un nouveau champ de recherche, et a amené à la parution d'un nombre important d'articles utilisant ces données, et notamment concernant le DARP. Une particularité de ces articles est qu'il manipule des instances ayant un nombre de requêtes sensiblement plus élevé que celle de [Solomon, 1987].

Par exemple, [Bertsimas et al., 2019] est un article sur le VRPTW dynamique avec des instances comptant plusieurs milliers de taxis et 25000 requêtes par heure, extrait de données réelles à Manhattan. Pour résoudre un DARP dynamique sans réservation, [Lioris and

3. Partager un véhicule sans échange marchand est soit du covoiturage, soit de l'auto-stop.

Cohen, 2016] proposent une heuristique dans Paris (ville simplifiée en un graphe à 288 sommets), avec 15000 clients par heure. [Burghout et al., 2015] traitent 270000 requêtes à Stockholm (ville simplifiée en un graphe à 421 sommets) pour un problème similaire, [Tsao et al., 2018] traitent un million de requêtes (66 régions), [Alonso-Mora et al., 2017] proposent un algorithme capable de traiter 4 requêtes par seconde à Manhattan. [Gurumurthy and Kockelman, 2018] traitent 6.2 millions de trajets à Orlando (Floride). [Martinez et al., 2014] traite 21000 requêtes et requiert 36 heures de calcul.

Résolution du DARP

Pour résoudre le DARP statique, on retrouve les méthodes utilisées pour le CVRPTW, comme la génération de colonne [Parragh and Schmid, 2013]. Des heuristiques de recherche de voisinage sont largement utilisées, notamment la recherche à voisinage variable [Parragh et al., 2010, Muelas et al., 2015].

Pour résoudre le DARP dynamique, les approches utilisées sont fondamentalement différentes. Dans les articles sur les problèmes dynamiques mentionnés deux paragraphes plus haut, les méthodes utilisées pour générer les emplois du temps des véhicules sont des algorithmes gloutons myopes, sans réservation dans le cas dynamique. L'objectif dans ces articles ne semble pas être d'établir les performances d'une méthode pour résoudre un problème donné comparée à d'autres, mais plutôt de déterminer le potentiel d'une activité ou d'un système. Une des conclusions de [Burghout et al., 2015] est par exemple qu'un système de véhicules autonomes partagés pourrait réduire de 95% le nombre de véhicules utilisés sous certaines conditions. Dans le cas de [Martinez et al., 2014], l'objectif est de simuler un système, et utilise donc un modèle agent, pour représenter naturellement chaque preneur de décision.

Au final, la recherche d'heuristiques fortement scalables pour le DARP statique ou dynamique ne semble pas être très développée. On peut tout de même citer pour le DARP statique [Muelas et al., 2015] qui évaluent une VNS (Variable Neighbourhood Search) qui partitionne une ville sur des instances ayant jusqu'à 16 000 requêtes. Les instances considérées sont naturellement adaptées au clustering, dans la mesure où l'origine ou la destination d'un client doit être comprise parmi un très faible nombre de points. Dans le sous-problème de "concert de musique" par exemple, tous les clients ont la même origine. On peut aussi citer [Agatz et al., 2011] qui regroupe les origines et destinations par comtés à Atlanta (graphe à 10 sommets), sur des instances à 3 millions de requêtes.

2.2 Problèmes récents

La figure 2.5 reprend la construction qui a mené du TSP au DARP. Pour établir la relation entre le DARP et le problème étudié dans cette thèse, qu'on appelle le PCE-ADARP, il ne manque que deux étapes, qui sont 1) Présenter les variantes récemment ajoutées à la famille du DARP pour prendre en compte les questions associées au développement technologique des véhicules, et 2) Présenter les spécificités de notre variante vis-à-vis de celles-ci.

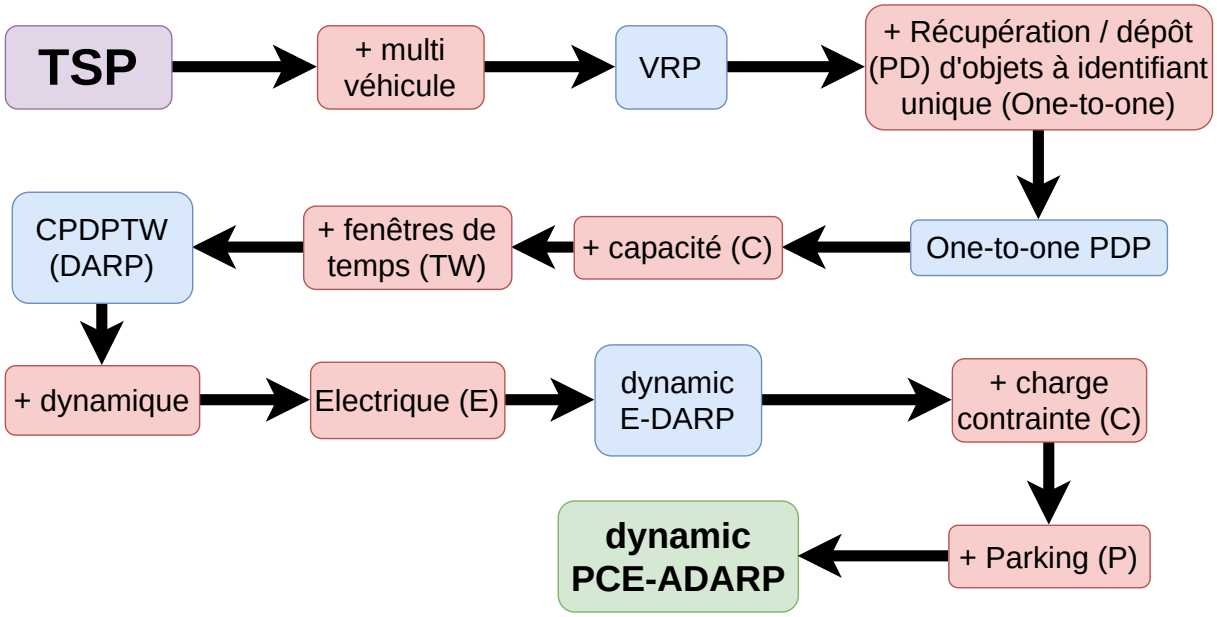


Figure 2.5 – *Successions des modifications nécessaires pour aller du TSP jusqu'au PCE-ADARP dynamique.*

2.2.1 Électrification

On commence par ajouter au DARP des contraintes liées à l'électrification. La batterie de chaque véhicule commence avec une certaine quantité d'énergie stockée, et cette énergie est dépensée à chaque déplacement. Ils ne doivent pas se retrouver avec une quantité d'énergie négative. Dans le graphe, en plus des sommets de récupération et de dépose d'usager, des sommets de charge sont ajoutés. Les véhicules peuvent s'y rendre pour se recharger.

La littérature sur l'électrification des véhicules est déjà riche, bien que récente. Il existe des articles liés mais pas tout à fait sur ce sujet datant des années 2000. À cette époque, le développement des piles à hydrogène a amené à un développement des problématiques liées aux véhicules à carburant alternatif. Il existe également des articles sur les transports en commun électriques, et notamment les bus électriques.

Un article largement cité dans le domaine des véhicules électriques est [Erdoğan and Miller-Hooks, 2012] sur le *Green-Vehicle Routing Problem*, qui est une extension du VRP dans laquelle les véhicules ont une autonomie limitée mais peuvent se recharger, et seules des charges complètes sont possibles. Bien que récent, il a été cité plus de 400 fois à ce jour (d'après *sciencedirect*), ce qui en fait un article fondamental dans ce domaine. Le G-VRP n'est pas spécifiquement dédié aux véhicules électriques, mais plutôt aux véhicules à biocarburant, avec une infrastructure de remplissage limitée. Dans les articles dédiés aux véhicules électriques, c'est plutôt le terme *Electric* qui est utilisé, plutôt que *Green*. On parle donc de E-DARP pour désigner un DARP avec des contraintes de batterie.

L'électrification des véhicules est peu évoquée dans les revues de la littérature du DARP mentionnés ([Cordeau and Laporte, 2007, Agatz et al., 2012, Ho et al., 2018]), et présentée plutôt comme une question future, sans article associé.

Dans certains articles, comme [Lin et al., 2016] ou [Erdoğan and Miller-Hooks, 2012], seules des charges complètes sont autorisées. Cela étant, recharger une voiture électrique

est un processus qui peut être très long. Pour cette raison, il est courant d'autoriser les charges partielles [Felipe et al., 2014, Merve and Bülent, 2016, Guillou et al., 2011, Ha et al., 2013, Ha et al., 2014], en particulier dans les articles les plus récents.

Le Green-VRP a été enrichi de différentes manières. [M. Sweda et al., 2017] rendent l'accès à une station de recharge incertain. [Montoya et al., 2017] incorporent des charges avec une vitesse non-linéaire, pour prendre en compte le fait que, au-delà de 80% de charge, les batteries lithium-ion se chargent de plus en plus lentement. [Lin et al., 2016] font varier la consommation énergétique en fonction du poids du contenu du véhicule. [Pelletier et al., 2018] incluent des particularités du transport de marchandise.

Le premier article dédié à la gestion d'une flotte de taxis électriques partagés est sans doute [Gacias and Meunier, 2012], qui présente un modèle pour un E-VRPTW permettant des charges partielles. Cet article est cependant très peu cité. Le point de départ pour la recherche sur la gestion d'une telle flotte serait plutôt [Schneider et al., 2014], qui présentent un modèle similaire, et est cité plus de 300 fois.

Simultanément, plusieurs articles enrichissent le Green-VRP en permettant des véhicules ou chargeurs hétérogènes. [Felipe et al., 2014] l'étendent en ayant des chargeurs de différentes vitesses, des charges partielles, des véhicules mixtes avec véhicules essence, et un niveau de détail élevé pour les calculs de consommation énergétique.

[Goeke and Schneider, 2015] permet des flottes hétérogènes, partiellement composées de véhicules essence. Les trajets que des taxis doivent effectuer ont des longueurs très variables. Avoir un faible pourcentage de véhicules avec une forte capacité pourrait être plus adapté qu'une flotte parfaitement homogène.

Un certain nombre d'articles proposent des approches très simples pour la gestion des taxis, et se concentrent plus sur des questions de nature économique, ou des comparaisons avec des systèmes plus traditionnels. [Li et al., 2016] évaluent ainsi comment la diversité des trajets des taxis influe sur la taille idéale des batteries. [Delos Reyes et al., 2016] évaluent l'impact de l'hiver sur la portée des véhicules (distance parcourue par plein). En effet, la réaction chimique des batteries lithium-ion est moins efficace à faible température, et le chauffage de l'habitacle représente une consommation énergétique non-négligeable ; une portée réduite de 50% en hiver n'est pas exclue. Étonnamment, les articles traitant de véhicules électriques autonomes ne mentionnent généralement pas la consommation passive des véhicules, ce qui est particulièrement gênant dans le cas de véhicules autonomes, par nature équipés de multiples capteurs et d'un ordinateur puissant.

Aux États-Unis, le système privilégié est celui des chargeurs rapides, capable de remplir une batterie en quelques minutes, et une taille des batteries importantes. Technologiquement, l'objectif est d'obtenir des voitures électriques similaires à des voitures essences du point de vue de la durée nécessaire pour faire le plein et la portée. [Bauer et al., 2018] évaluent les performances économiques des taxis autonomes électriques à Manhattan, [Hu et al., 2018] déterminent quels trajets sont possibles pour une flotte à New-York, compte tenu de la taille des batteries de ses véhicules. [Fraile-Ardanuy et al., 2018] font de même à San-Francisco.

[Liang et al., 2016] étudient le E-VRPTW dans le cas particulier de taxis partant ou allant vers une gare.

En Chine, l'échange de batteries est beaucoup plus courant. Ce système consiste à créer des voitures dont la batterie est facilement remplaçable, de sorte que retirer une batterie vide et la remplacer par une pleine se fait en quelques minutes (voire moins d'une minute).

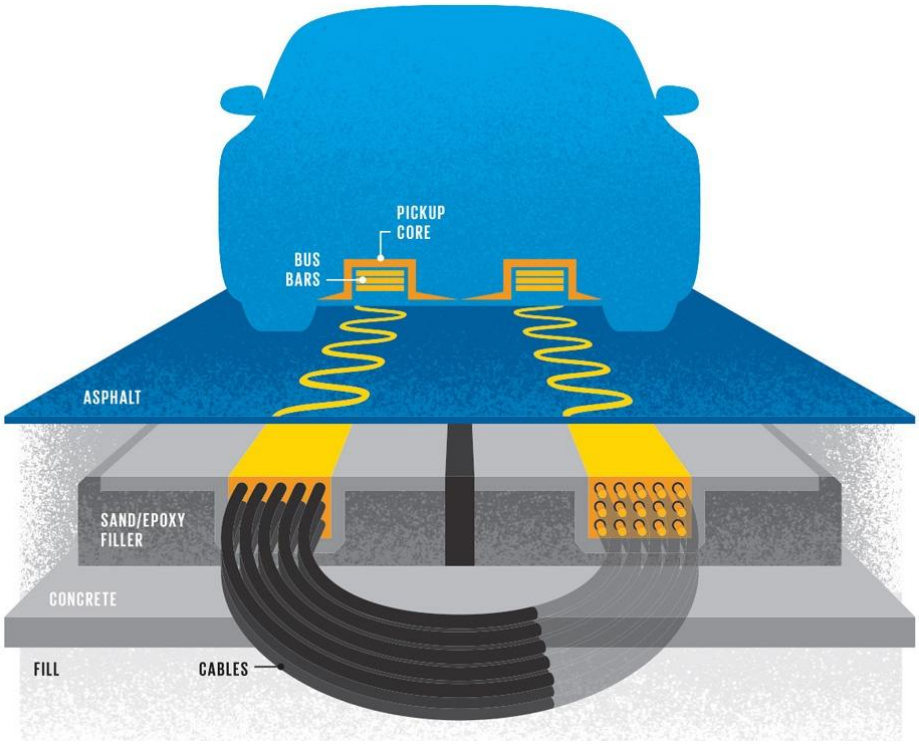


Figure 2.6 – *Recharge par induction. Des bobines sont placées sous la route, et de l'énergie est transmise sans contact par électromagnétisme au véhicule circulant dessus.*

Ce système peut complémenter un système de recharge lente efficacement sous certaines conditions. La compagnie *Better Place* a échoué en 2012 à créer un réseau d'échange de batteries en Israël, faute d'un manque d'usagers ; tandis que ce système fonctionne bien en Chine, où l'on trouve de nombreux véhicules électriques utilisant le même modèle de batterie. Les articles étudiant cette technologie sont donc presque exclusivement chinois. [Liao et al., 2016] comparent l'efficacité d'un système de recharge rapide des batteries à un système d'échange de batterie, et [Cao et al., 2018] étudient un système d'échange de batteries avec réservation. Plusieurs articles sont dédiés au placement optimal de stations de recharge, comme [Hiermann et al., 2016].

Une autre technologie de recharge est la recharge par induction. Elle permet à un véhicule de se recharger sans utiliser de cable ou de borne de recharge (*cf.* figure 2.6). Cette technologie permet aux véhicules de se recharger en continu, et notamment d'effectuer des micro-charges aux stops et feux rouges. La recharge par induction souffre cependant de plusieurs d'effet qui limite sa génération. On peut notamment citer un taux de perte d'énergie important, ou une usure plus rapide de la batterie [zzz15, 15]. La recharge par induction peut être intégrée à un E-VRPTW simplement en changeant la consommation énergétique associée à certains arcs, et ne change donc pas fondamentalement le problème, sauf à permettre des déplacements avec gain d'énergie.

Mêler partage de course et électrification des véhicules est un sujet récent, et étudié notamment dans [Bongiovanni et al., 2018] pour le E-ADARP dynamique et [Bongiovanni et al., 2019] pour le E-ADARP statique.

Résolution des problèmes d'électrification

Dans [Erdoğan and Miller-Hooks, 2012], deux heuristiques sont proposées pour résoudre le G-VRP. La première, appelée MCWS, est une extension du "Clarke and Wright Saving Algorithm" de [Clarke and Wright, 1964]. Dans cette heuristique, un cycle "depot - v - depot" est créé pour tout sommet v, puis ces cycles sont fusionnés progressivement, jusqu'à obtenir un nombre de cycles inférieur/égal au nombre de véhicules, ou échouer, dans le cas où des contraintes seraient violées quelle que soit la fusion effectuée. Dans le cas d'un échec, des véhicules supplémentaires sont ajoutés. L'objectif dans cet article n'est pas "traiter le plus de clients avec un nombre de taxis donné" mais "traiter tous les clients avec le minimum de taxis". La seconde heuristique utilisée dans [Erdoğan and Miller-Hooks, 2012] est appelée DBCA, et est basée sur le "Density-Based Clustering of Application with Noise" de [Ester et al., 1996]. Cette seconde approche consiste à grouper les points de passage et à faire traiter les points d'un même groupe par un même véhicule. Cette approche est peu adaptée pour les variantes du PDP car l'origine et la destination d'un client peuvent être très éloignées l'une de l'autre, donc il n'est pas nécessairement bon d'attribuer deux clients à l'origine proche à un même véhicule.

La méthode utilisée dans [Schneider et al., 2014] pour résoudre le E-VRPTW avec charge partielles est la suivante. Une solution initiale est générée avec un algorithme glouton. Une fois cette solution initiale générée, elle est améliorée avec une recherche de voisinage qui mélange recuit simulé, recherche tabou, et recherche à voisinage variable (VNS : Variable Neighbourhood Search). Ce principe général a été largement réutilisé dans les années suivantes.

[Merve and Bülent, 2016] proposent une approche différente pour le E-VRPTW avec charges partielles. Il s'agit encore une fois d'une heuristique de recherche locale, qui consiste principalement à retirer puis insérer des requêtes (approche *destroy-and-repair*) avec une approche de type recuit simulé. Une particularité de l'approche proposée est qu'il s'agit d'une ALNS (Adaptative Large Neighbourhood Search). Dans [Felipe et al., 2014], qui, comme on l'a mentionné, enriche le E-VRPTW de diverses manières, on retrouve également un ALNS destroy-and-repair avec un recuit simulé, et c'est encore une telle approche qui est utilisée pour [Goeke and Schneider, 2015].

Pour résoudre le E-ADARP statique, [Bongiovanni et al., 2019] utilise un Branch-&-cut avec une formulation du problème en 2-index, et a des difficultés à traiter des instances [Solomon, 1987] avec 50 clients en moins de 2 heures.

Pour résoudre le E-ADARP dynamique, [Bongiovanni et al., 2018] présente des résultats préliminaires d'une heuristique en deux temps (insertion simple de requête, amélioration de la solution via Machine-Learning). [Desaulniers et al., 2016] utilise aussi une approche Branch-&-cut, et atteint les 100 clients pour un E-VRPTW. Ce type d'approche est beaucoup utilisé pour le DARP, comme [Pisinger and Ropke, 2005].

2.2.2 Autonomisation

Ajouter une notion d'autonomie des véhicules au DARP, et ainsi le transformer en ADARP (A = Autonomous) ne complexifie pas le problème, et c'est même plutôt le contraire. Les chauffeurs humains posent un ensemble de conditions pour accepter de travailler. Dans le cas d'un service tel que Uber par exemple, le gestionnaire (la compagnie Uber) s'engage

auprès des chauffeurs, pour s'assurer que les courses sont réparties équitablement entre eux, et ainsi que les gains d'argent soient répartis plutôt équitablement. Il arrive aussi que les chauffeurs disposent d'un certain degré d'autonomie que n'ont pas, paradoxalement, les véhicules autonomes.

Ce sont ces raisons qui font que, depuis environ l'an 2000, les modèles agents sont souvent utilisés pour modéliser les flottes de taxis conventionnels [Ronald et al., 2015]. Un modèle agent est un modèle à intelligence distribuée dans lequel chaque taxi (agent) peut faire des choix, et cherche à maximiser son profit individuel. Lorsqu'un tel modèle est utilisé, le but est de trouver un système de récompense des agents tel que, si chaque agent est rationnel, le profit global est maximisé. Il est aussi courant d'ajouter des objectifs annexes, par exemple, essayer d'avoir la répartition du profit jugée la plus équitable.

Un ADARP pourrait donc être grossièrement défini comme un DARP pour lequel une gestion centralisée des véhicules est explicitement permise, et n'ayant pas certaines contraintes auxiliaires (comme un dépôt spécifique auquel revenir à la fin de la journée pour chaque chauffeur). En pratique, le terme *autonomous* semble plus être utilisé pour attirer l'attention d'un public non-spécialiste que pour indiquer une propriété théorique précise du problème considéré.

2.3 Le problème considéré dans cette thèse

Notre problème est une extension de E-ADARP. Comme il a été mentionné à plusieurs reprises dans ce chapitre, il existe de nombreuses variantes du VRP, du PDP et du DARP. Une présentation exhaustive des détails de notre modèle est présentée dans les chapitres 3 et 4.

Présentons ici les deux caractéristiques additionnelles principales de notre problème, qui sont 1) la disponibilité des chargeurs et 2) la recherche de place de stationnement.

2.3.1 Charge contrainte

Dans les articles traitant du E-ADARP, la disponibilité des chargeurs est ignorée. Dans ces articles, un point de recharge représente une station de recharge de capacité infinie, comme dans [Bongiovanni et al., 2019]. En d'autres termes, les chargeurs ne constituent pas une ressource partagée, et les taxis ne se gênent donc pas entre eux pour se recharger.

Quand on en cherche la raison, on constate que les premiers articles sur la recharge des véhicules électriques sont inspirés d'articles sur le plein de véhicule essence, par exemple de camions pour des trajets longs dans des régions ayant peu de stations. C'est vraisemblablement pour cette raison que les premiers articles sur la recharge de véhicules électriques supposent que les charges sont complètes et que la disponibilité d'un chargeur est permanente et instantanée.

L'infrastructure de recharge rapide est actuellement sous-dimensionnée. De plus, il paraît peu probable que l'infrastructure de recharge soit réservée pour l'usage exclusif des taxis, et une flotte de véhicules électriques devrait avoir une certaine résilience face à une disruption du fonctionnement de l'infrastructure de recharge. Il a donc paru intéressant de supposer que les chargeurs ne sont pas toujours disponibles.

De prime abord, la gestion de la recharge de taxis peut sembler simple. Le jour, les taxis sont fortement sollicités, et transportent des usagers. La nuit, ils sont peu sollicités, et en profite pour se recharger. La réalité va dans ce sens, dans une certaine mesure, comme le montre la figure 2.7.

Pour qu'une recharge exclusivement nocturne fonctionne, trois conditions doivent être réunies. Premièrement, l'infrastructure de recharge doit être suffisante pour le permettre, deuxièmement, les taxis doivent avoir suffisamment de batterie pour rouler toute la journée, et troisièmement, la proportion de trajets la nuit ne doit pas croître.

Les résultats présentés par [Hu et al., 2018, Bauer et al., 2018, Fraile-Ardanuy et al., 2018] indiquent que des recharges en cours de journée sont actuellement nécessaires, et que l'infrastructure actuelle est insuffisante. Des taxis ayant suffisamment de batterie pour tenir une journée entière pourrait être utilisés, mais leur batterie serait lourde, augmentant sensiblement le coût des déplacements. On peut aussi mentionner la baisse sensible du coût des trajets nocturnes et les opportunités pour les livraisons [Taefi, 2016, Mommens et al., 2018, Ukkusuri et al., 2015, Morganti et al., 2014], qui devrait augmenter l'usage de ces véhicules en heure creuse.

Ces raisons, en plus de celles données dans le chapitre précédent, font qu'il est important de considérer la disponibilité de l'infrastructure de recharge : Quand les chargeurs sont ils utilisables, et combien de temps faut il faire la queue en particulier.

2.3.2 Stationnement

La seconde caractéristique importante est la prise en compte du stationnement dans le calcul de la consommation énergétique.

D'une part, le stationnement peut représenter une part importante du trafic d'une ville. Il est difficile d'évaluer la proportion exacte, qui peut beaucoup varier en fonction de l'heure et de la ville, mais il est courant de lire dans des revues non-scientifiques que cette recherche représente entre 5 et 30% du trafic en ville. Il paraît donc pertinent de supposer que cette recherche affecte également les taxis autonomes.

D'autre part, il est attendu que les taxis autonomes aient plus de mal à s'arrêter que les chauffeurs de taxis traditionnels. Ces derniers peuvent inventer des places de stationnement de courte durée sans danger ni gêne de circulation majeure en utilisant du bon sens et de l'expérience, tandis que les taxis autonomes devront sans doute suivre à la lettre les règles légales.

Un inventaire exhaustif des lieux de stationnement est à ce jour inexistant. [Chester et al., 2010] estiment par exemple que le nombre de places de stationnement sur la voirie aux États-Unis se situe entre 105 millions et 2 milliards. Cette estimation est inspirée de [Delucchi, 1997] à l'aide des réglementations de construction américaine ; de [Shoup, 2005, Davis et al., 2010], qui utilisent des enquêtes ; et de [Akbari et al., 2003] qui utilisent des images satellites, entre autres. Toutes ces méthodes sont indirectes, limitées à une zone géographique, et ces articles utilisent différentes définitions de place de parking. Le même constat peut être fait dans les autres régions du monde.

La principale question associée au stationnement en voirie dans cette thèse est le temps de recherche nécessaire pour trouver une place. [Lefauconnier and Gantelet, 2005] conclut que le temps de recherche en métropole est généralement de l'ordre de 8 minutes. Des études avec une granularité plus fine ont été menées dans diverses villes, comme [Belloche,

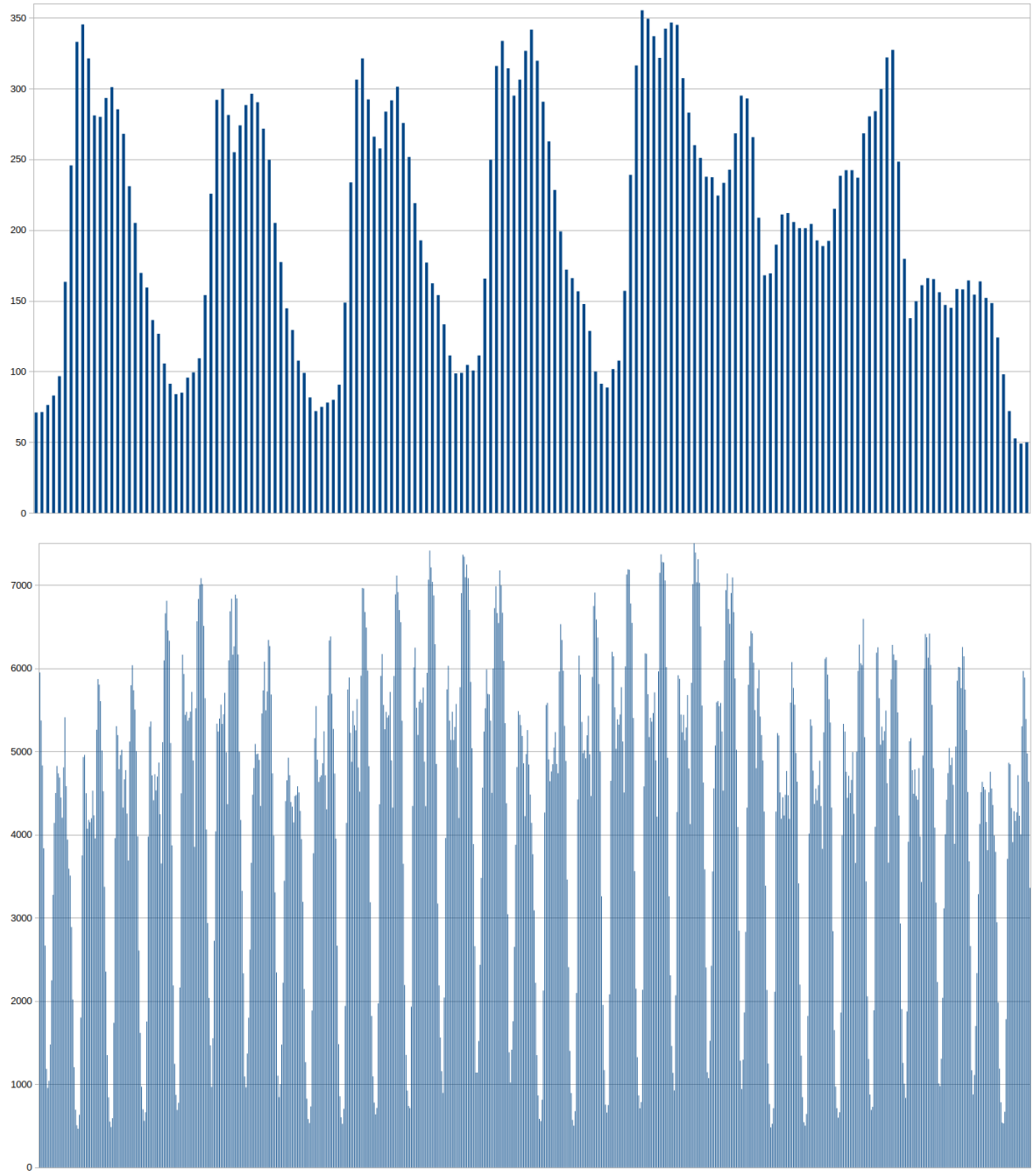


Figure 2.7 – *Clients traités chaque heure par les taxis de deux villes. En haut, à Porto durant une semaine, en commençant un lundi à 00 :00. En bas, à New-York pour le mois d'avril 2012, en commençant un dimanche à 00 :00.*

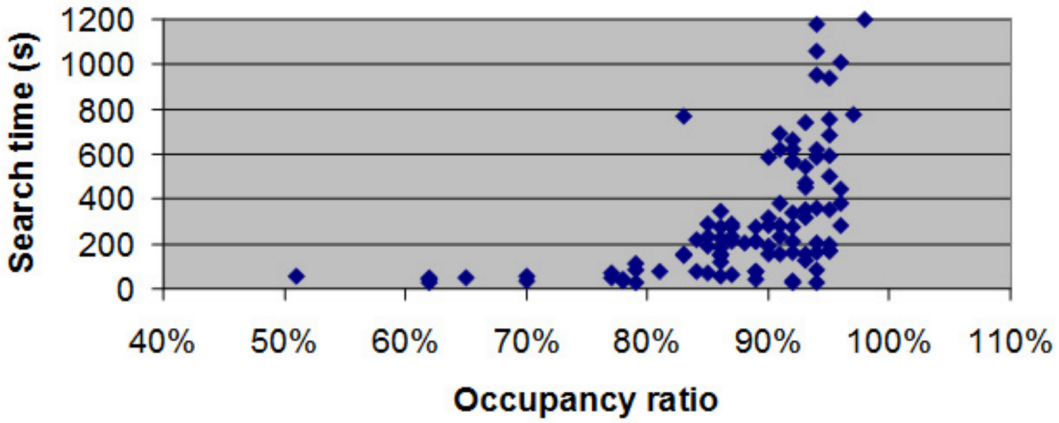


Figure 2.8 – *Temps de recherche pour trouver une place de stationnement à Lyon, en fonction du taux d'occupation des places de stationnement dans la zone où une place est cherchée.*

2015] qui présente le temps de recherche dans 10 quartiers de Lyon. La méthodologie utilisée dans cet article est la suivante, qui est par ailleurs standardisée et utilisée internationalement. Une expérimentation est menée, dans laquelle les expérimentateurs tentent de garer leur véhicule dans un quartier donné. Le résultat théorique le plus intéressant obtenu avec ces expérimentations est sans doute celui présenté dans [Polak and Axhausen, 1990], qui est que le temps nécessaire pour trouver une place est fortement corrélé au taux d'occupation de ces places, et explose lorsqu'il dépasse 80% (voir figure 2.8). Ce résultat a eu une influence significative pour la tarification de zones de stationnement [4], car si le taux d'occupation des places de stationnement dans une zone dépasse 80%, il est sans doute judicieux d'y placer des parcmètres, ou d'augmenter le coût du stationnement.

Une des limites de ce type d'étude est que le temps de recherche dans une zone n'est pas corrélé à l'heure de la journée ou au jour de la semaine. De fait, si une mairie apprend qu'il est difficile de trouver une place le week-end, elle ne va probablement mener une étude que sur le week-end, aux heures les plus chargées. L'évolution dans le temps de l'utilisation d'un réseau de stationnement typique n'est ainsi pas connu à ce jour. [Caicedo et al., 2012] présentent le taux d'occupation d'un parking souterrain payant au cours du temps, mais ce résultat n'est probablement pas utilisable pour déterminer le taux d'occupation de places sur voirie d'un quartier résidentiel ou commercial typique. Alors que les parkings gratuits sont peu étudiés, les données relatives aux parkings payants sont généralement privées et inaccessibles. On peut aussi noter que la méthodologie employée dans [Caicedo et al., 2012] imposent une limite sur le temps. Dans [Belloche, 2015], une recherche de place est automatiquement considérée comme réussie au bout de 20 minutes, et le temps maximum de recherche observé dans la moitié des quartiers est de 20 minutes.

Des études ont également été menées concernant la meilleure stratégie à employer pour rechercher une place, comme [Houissa et al., 2017]. Bien que ce type d'algorithme soit utile pour des véhicules autonomes, les calculs qu'ils imposent sont chronophages, et semblent peu adaptés à notre problème et des instances de grande taille, pour lequel une approche plus macroscopique est préférée.

4. en particulier en Europe

2.3.3 Résumé du PCE-ADARP

Le problème étudié dans cette thèse est nommé PCE-ADARP (Parking & Constrained Electrification in an Autonomous Dial-A-Ride Problem). Il s'agit, en résumé, du DARP présenté dans la section 2.1.4, auquel il faut ajouter que les véhicules ont une batterie limitée, qu'ils peuvent recharger à des chargeurs. L'accès aux chargeurs est contraint de trois manières. Plusieurs véhicules ne peuvent pas se recharger au même chargeur en même temps, les chargeurs ne sont pas toujours disponibles, et une attente est généralement nécessaire avant d'accéder à un chargeur. Un ajout supplémentaire est que le calcul de la consommation électrique prend en considération la recherche de place de stationnement.

Ayant présenté dans les grandes lignes les caractéristiques du problème et sa relation aux problèmes déjà étudiés par la communauté scientifique, présentons maintenant plus en détails les choix de modélisation ayant mené à l'élaboration d'un modèle mathématique.

Chapitre 3

Élaboration du modèle théorique

Dans ce chapitre, nous présentons la manière dont un problème pratique dans une ville est transformé en un problème théorique. La modélisation exacte du problème est présentée dans le chapitre 4. Une part importante du présent chapitre est consacrée aux choix de modélisation et à leur justification.

Un aspect important du modèle est qu'il est parfaitement déterministe, dans la mesure où il ne contient aucune composante stochastique. Nous verrons par exemple dans la suite de la présentation du modèle que la durée d'un trajet ou la disponibilité d'un chargeur sont prédéterminées et connues du gestionnaire. La seule inconnue pour le gestionnaire est l'ensemble des requêtes qui n'ont pas encore été formulées dans le cas dynamique.

3.1 Données d'entrées

Le modèle est en temps discret, tel que 1 instant est équivalent à 1 seconde. On appelle t_0 l'instant à partir duquel les taxis peuvent commencer à bouger et se charger, et t_{fin} l'instant auquel ils doivent s'arrêter.

3.1.1 La ville et son graphe

Dans la suite de ce mémoire, le terme *rue* désigne un ensemble de voies de circulations utilisables par des taxis, allant dans le même sens et comprises entre deux intersections. Une *intersection* est une zone de connexion entre au moins trois rues [1]. Tout lieu circulable par des taxis (hors parking) est soit une rue soit une intersection.

Le terme *ville* désigne l'ensemble des rues et intersections dans un périmètre donné.

Une ville peut être représentée par un graphe dans lequel chaque sommet représente une rue, et chaque arc d'un sommet a vers un sommet b signifie qu'il est possible d'atteindre b à partir de a à une intersection. La figure 3.1 montre un exemple de ce graphe.

Chaque sommet a des coordonnées GPS, qui sont celles du dernier mètre de la voie de circulation que le sommet représente. Par exemple, dans la figure 3.1, les coordonnées du sommet C sont au niveau de la ligne de stop, sur la file de droite.

Dans la suite de ce mémoire, on appelle ce graphe le *graphe de la ville*.

1. Une portion d'autoroute entre une entrée et la sortie suivante est donc une rue

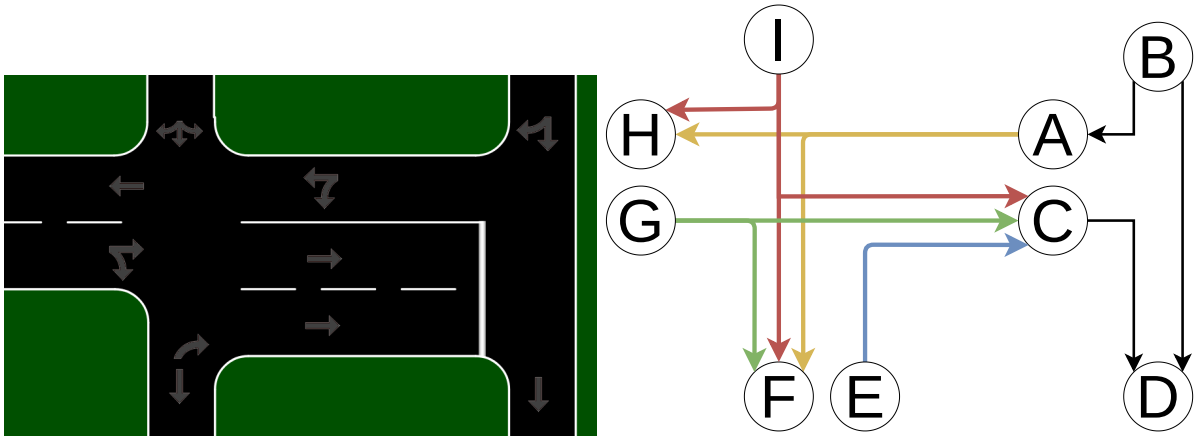


Figure 3.1 – *A gauche, représentation d'une portion de ville. A droite, le graphe correspondant. L'arc (I,H) représente ainsi le fait qu'une personne arrivant de la rue en haut à gauche peut aller vers la rue de gauche. On peut noter que les sommets E et F représentent chacun un sens de circulation d'une même voie à double-sens et que le sommet C ne représente pas une voie de circulation, mais deux.*

Propriétés des arcs et déplacements

Notre modèle n'a pas vocation à faire de la micro-simulation, on ne s'intéresse pas à la manière exacte dont un taxi se déplace (dépassement, trafic en accordéon, attente à un feu rouge, ...).

Chaque arc a une longueur (en mètres) et une vitesse moyenne d'usage (en mètre par seconde). Lorsqu'un véhicule emprunte un arc, il se déplace à une vitesse constante et déterministe qui est la vitesse moyenne d'usage de cet arc.

A chaque instant, la ville a une valeur de congestion, qui détermine à quel point il est difficile de circuler. Tout trajet commençant à un instant t voit sa durée multipliée par la valeur de congestion de la ville à cet instant. Par exemple, dans le cadre d'un trajet hypothétique de 50km, la durée de traversée de la dernière rue utilise la valeur de congestion en vigueur au départ de la première rue. On appelle *durée de base* la durée d'un trajet avant que ne soit pris en compte le multiplicateur de congestion, autrement dit, si le multiplicateur à l'instant t était 1. Dans le cas d'une course partagée, qui contient des détours, le déplacement entre chaque arrêt est considéré comme un trajet différent en ce qui concerne la congestion.

Dans la suite de cette thèse, "plus court chemin" signifie "chemin ayant la plus petite durée de base". Il ne s'agit pas nécessairement du plus court chemin en distance. Parmi les chemins de même durée de base, origine, et destination, l'un d'eux est arbitrairement considéré "plus court" que les autres, donc le plus court chemin d'un sommet a vers un sommet b est unique, et ne varie pas dans le temps. Lorsqu'un taxi se déplace, il utilise toujours le plus court chemin de son origine vers sa destination.

Cette modélisation particulière des déplacements et de la congestion a été choisie afin de réduire au maximum l'impact des calculs de plus courts chemins dans les méthodes que nous allons présenter dans les chapitres 6 et 7. En effet :

— Puisqu'il n'existe qu'un trajet possible de a vers b, il est possible de précalculer le plus court chemin de n'importe quelle rue vers n'importe quelle rue à l'aide de l'algorithme de Floyd-Warshall.

— Puisque le multiplicateur de congestion dépend uniquement de l'instant de départ d'un

trajet, et qu'il est possible de précalculer la durée de base d'un trajet, alors calculer la durée effective d'un trajet peut se faire en temps constant.

— Enfin, dans deux instances utilisant le même graphe de la ville, les plus courts chemins ne changent pas, ce qui permet de réutiliser le résultat d'un précalcul sur plusieurs instances.

Propriétés générales du graphe

Pour garantir que les taxis peuvent circuler, le graphe de la ville doit être fortement connexe. On ne peut pas accepter qu'un taxi se retrouve bloqué dans un sommet qui n'a pas d'arc sortant, ou qu'un chargeur soit inaccessible (pas d'arc entrant). Nous verrons dans le chapitre 8 comment un graphe fortement connexe est généré à partir des données accessibles.

Ce graphe est de faible densité, et dans les cas concrets, plutôt de grande taille (milliers, voire dizaines de milliers de sommets/rues). Il n'est pas planaire.

Ayant présenté le graphe de la ville, on peut maintenant présenter les objets associés à ce graphe : taxis, chargeurs, et requêtes.

3.1.2 Taxis

Chaque taxi a un nombre maximal de passagers, un point de départ (sommet du graphe de la ville), une capacité maximale de batterie, un niveau de charge initial de cette batterie, un niveau minimal final de charge et des paramètres de consommation électrique lorsqu'il se déplace.

Dans la réalité, la consommation énergétique dépend de nombreux critères. On a mentionné la température, mais la vitesse a une importance aussi grande que pour les véhicules thermiques, de même que la charge du véhicule ou l'inclinaison de la route [Fraile-Ardanuy et al., 2018].

Dans notre modèle, la consommation énergétique d'un taxi à chaque instant est la somme de deux composantes.

1. A chaque instant, un taxi consomme de l'énergie en fonction de sa vitesse. Cette part de la consommation n'est pas strictement proportionnelle à la vitesse (Par exemple, à 100km/h, la consommation par mètre est plus élevée qu'à 20km/h).
2. A chaque instant, les systèmes auxiliaires consomment de l'énergie. Cette consommation regroupe celles de l'ordinateur de bord, des capteurs, de la climatisation, et plus généralement de tous les composants d'un véhicule autonome ne servant pas à faire tourner les roues.

Lorsqu'un taxi est dans un sommet v et qu'il veut s'arrêter, il doit trouver une place de stationnement. La recherche d'une place a une durée fixe inspirée de [Lefauconnier and Gantelet, 2005], et le taxi continue de consommer de l'énergie pendant cette recherche. Les mouvements précis effectués par le taxi durant cette recherche ne sont pas modélisés. A la fin de cette recherche, il trouve toujours une place dans le sommet v (la rue v).

[Billhardt et al., 2017] présentent un système de gestion d'une flotte ouverte. Une flotte ouverte y est définie comme une flotte de véhicules détenus par des particuliers, qui entrent

et quittent la flotte librement, et sont sujets à peu de contraintes : Il s'agit du type de flotte gérée par Uber. On peut imaginer que les véhicules autonomes vont développer ce type d'activité, en permettant aux particuliers de louer leur véhicule lorsqu'ils n'en ont pas l'usage, pour quelques heures ou quelques jours. Ces véhicules ajoutent un certain nombre de contraintes pour un gestionnaire de flotte, comme une flotte variable en taille, composée de véhicules hétérogènes, peu prévisibles, et à l'usage restreint [2]. Bien qu'il s'agisse d'un problème intéressant, nous avons choisi de modéliser une flotte fermée, plutôt homogène, pour simplifier le problème et l'analyse des résultats.

3.1.3 Chargeurs

Chaque chargeur a un emplacement (sommet du graphe de la ville) et une puissance (vitesse de charge). Une caractéristique importante des chargeurs est leur disponibilité, qui est limitée de trois manières.

Premièrement, on suppose que chaque chargeur ne peut être utilisé que par un taxi à la fois.

Deuxièmement, un chargeur n'est pas toujours utilisable par les taxis. Cela peut être dû à une panne, ou une restriction de la recharge aux heures de pics de consommation électrique. On suppose que le gestionnaire de la flotte prévoit sans erreur ces problèmes.

Troisièmement, on suppose que, dans le cas général, les chargeurs ne sont pas la propriété exclusive d'un propriétaire de flotte de taxis, et donc que des véhicules qui ne font pas partie de la flotte peuvent les utiliser. Par conséquent, lorsqu'un taxi arrive à un chargeur, il lui faut un certain temps (déterministe) pour accéder au chargeur, à la fois à cause des manœuvres nécessaires (connexion et déconnexion au chargeur), mais aussi pour laisser le temps à l'utilisateur précédent de libérer la place, et/ou de faire la queue.

La figure 3.2 montre comment les batteries de type lithium-ion (technologie utilisée par les véhicules électriques) se chargent au cours du temps en réalité. Dans notre modèle, en excluant le temps d'attente et de manœuvre, la batterie du taxi se remplit linéairement.

La variation d'énergie durant une charge dans notre modèle est donnée par la figure 3.3. Dans un premier temps, le véhicule perd de l'énergie (manœuvre et attente que le chargeur se libère). Au bout d'une durée a (voir figure 3.3), le taxi peut effectivement commencer à se charger. Cette durée est fixe, et commune à toutes les charges. Au bout d'une durée b, le taxi a regagné autant d'énergie qu'il en a dépensé pour pouvoir utiliser le chargeur. Au bout d'une durée c qui dépend du niveau de la batterie du taxi au début de la charge, la batterie du taxi est pleine.

3.1.4 Requêtes

Les taxis doivent traiter un ensemble de requêtes. Chaque requête à un nombre de passagers, une origine et une destination (sommets du graphe de la ville), et des caractéristiques temporelles.

2. Les propriétaires de véhicules peuvent par exemple demander que leur véhicule ait toujours au moins k% de batterie, ou ne s'écarte pas de plus de Xkm du point de départ du véhicule.

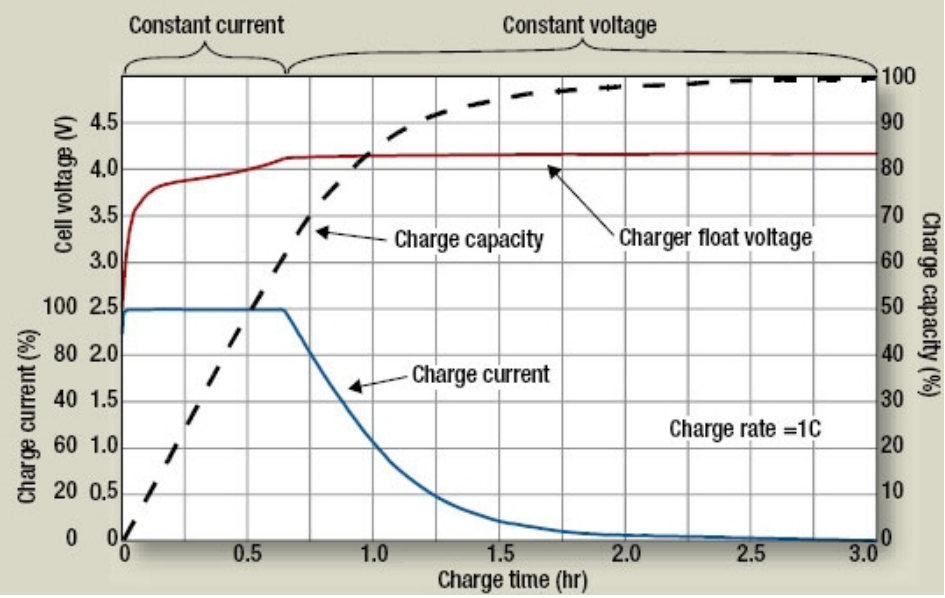


Figure 3.2 – *Le niveau de charge d'une batterie (pointillés noirs) au cours du temps. La charge est d'abord linéaire, avec un courant électrique constant (courbe bleue), puis lorsque la batterie atteint 60% de charge, la charge ralentit, avec un voltage qui devient constant (courbe rouge). Image de Cadex Electronics.*

Passagers

Dans la suite de cette thèse, une distinction importante est faite entre *passager* et *client*. Un passager est un individu physique. 1 passager occupe 1 place de taxi lorsqu'il monte dans ce taxi. Un client est une entité abstraite qui désigne, selon le contexte, soit l'ensemble indivisible des passagers d'une requête, soit l'émetteur d'une unique requête.

Autrement dit :

— Les requêtes sont indépendantes. Le client formulant une requête a est toujours supposé différent de celui qui formule une requête b. Il n'y a pas, entre autres, de paires de requêtes formant un aller-retour (tel que le retour ne peut pas avoir lieu si l'aller n'a

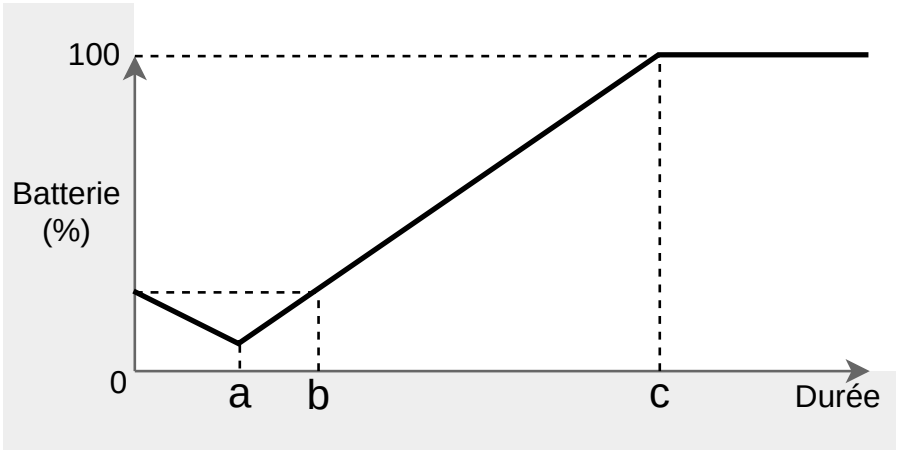


Figure 3.3 – *Variation d'énergie causée par une charge en fonction de sa durée.*

pas eu lieu).

— Un groupe de passagers associé à une requête est indivisible. Si l'un des passagers monte dans un taxi, tous les passagers de ce groupe doivent monter. De la même manière, si un passager descend, tous doivent descendre.

Il n'est pas possible non plus pour un taxi de déposer un client ailleurs qu'à sa destination, ou de transférer un client vers un autre taxi.

Dans la suite de ce mémoire, un client est dit *traité* ou *accepté* si et seulement si un taxi le récupère et l'amène à sa destination en respectant ses contraintes de temps et de partage. Il ne peut pas y avoir de client *partiellement* traité, ou partiellement satisfait. On peut aussi dire d'une requête qu'elle est traitée, avec le même sens. Une requête non-traitée est dite *rejetée*.

Valeur, Origine et destination

Chaque requête a une valeur qui est utilisée dans la fonction objective. Soit q une requête. Soit l la durée du plus court chemin entre l'origine et la destination de q en partant à l'heure idéale de départ de q (en prenant donc en compte la congestion à cette heure). Soit k le nombre de passagers de q. La valeur de q est $k \times l$.

Heure de formulation

Les caractéristiques temporelles des requêtes sont présentées dans la figure 3.4.

On distingue deux cas : le modèle statique (ou *offline*) dans lequel toutes les requêtes sont connues à l'avance, et le modèle dynamique (ou *online*), dans lequel les requêtes sont découvertes progressivement.

Dans le cas dynamique, une requête doit être acceptée ou rejetée de manière définitive en temps réel. Le gestionnaire n'a que quelques secondes pour se décider, il ne peut pas utiliser pour se décider des informations relatives à une requête qui arrive durant cet intervalle de quelques secondes, ou plus tard.

La finalité de la thèse est plutôt de développer des méthodes pour résoudre le cas dynamique du problème. L'étude du cas statique sert dans une large mesure à déterminer ce qu'une méthode de résolution faisant des prédictions parfaites des futures requêtes serait capable de faire, et ainsi de déterminer dans quelle mesure une méthode pour le cas dynamique pourrait être améliorée.

Il n'existe pas de contraintes sur l'heure de formulation de la requête, autre que le fait qu'elle a lieu avant l'heure idéale de départ du client. Dans nos expérimentations, les requêtes sont divisées en quatre groupes clairement distincts :

— Formulées avant t_0.

— Longtemps en avance. Quelques heures avant l'heure idéale de départ.

— Peu en avance. Quelques minutes avant l'heure idéale de départ.

— Au vol. L'heure de formulation est l'heure idéale de départ.

Nous nous sommes limités à cette granularité (4 groupes) principalement à cause d'un manque de données sur la réservation (*cf.* chapitre 8).

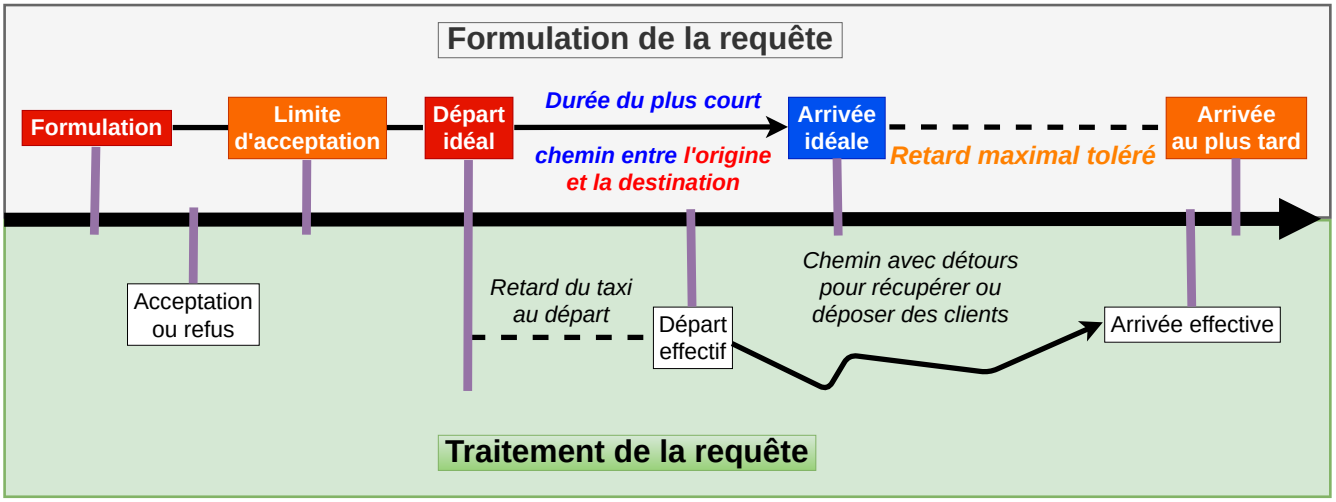


Figure 3.4 – *Une requête et son traitement dans le PCE-ADARP. Les informations en rouge sont celles données par le client, les informations en bleus sont inférées par le gestionnaire, et les informations orange sont principalement établies par des critères prédéterminés de qualité de service.*

Retard

Certains clients ne tolèrent aucun retard ; Ils indiquent une heure idéale de départ, et ne sont satisfaits que s'ils sont récupérés à cette heure précise, et que le taxi les amène à leur destination sans le moindre détour.

Pour les autres clients, deux types de retard sont distingués. Le premier est le retard au départ. Ce retard apparaît lorsqu'un taxi ne récupère pas un client à son heure idéale de départ. Le second est le retard accumulé alors que le client est dans le taxi.

Puisque les taxis sont partagés, un taxi peut aller récupérer un client alors qu'il en contient déjà un, à condition de respecter certaines contraintes (détaillées dans la sous-section consacrée au partage de course). Pour calculer ce retard, la durée du trajet pour amener le client à sa destination est comparée à la durée du plus court chemin de l'origine vers la destination du client.

Certains clients ne tolèrent aucun retard au départ. Il s'agit de clients qui ont formulé leur requête avant t_0 dans les instances dynamiques. L'hypothèse sous-jacente est que les gens qui réservent le plus tôt sont aussi les plus exigeants vis-à-vis du retard.

Les autres clients tolèrent un retard infini au départ. Aucun client n'accepte un retard infini à l'arrivée.

Le retard maximal total toléré par un client suit une fonction affine $ax + b$ dans laquelle a et b sont des constantes de qualité de service, communes à tous les clients, et x est la durée attendue de la course du client (plus court chemin de l'origine à la destination, en partant à

l'heure idéale de départ). Pour résumer les contraintes de temps des requêtes :

$$\begin{aligned}
&\text{heure de formulation} \\
&\quad \leq \text{heure idéale de départ} \\
&\quad \leq \begin{cases} \text{heure idéale d'arrivée} \\ \text{heure effective de départ} \end{cases} \\
&\quad \leq \text{heure effective d'arrivée} \\
&\quad \leq \text{heure maximale d'arrivée}
\end{aligned}$$

3.1.5 Partage de course

Le terme *course* est défini du point de vue des taxis plutôt que des clients. Plus explicitement, une course d'un taxi commence lorsqu'un taxi cesse d'être vide (sans passager), et se conclut dès que le taxi redevient vide. Tout ce qui se passe entre ces deux instants concernant le taxi est une seule et unique course. Si plus d'un client monte dans le taxi durant cette course, on dit qu'il s'agit d'une *course partagée*. Sinon, il s'agit d'une *course individuelle*.

Il est important de noter que, dans la suite de ce mémoire, ce qui détermine si une course est partagée ou individuelle n'est pas le nombre de passagers, mais de clients (de groupes).

Un taxi qui contient au moins un client ne peut faire que deux choses :

— Aller vers la destination d'un client qu'il contient, puis déposer ce client. Cette descente doit avoir lieu avant l'heure d'arrivée maximale du client.

— Aller vers l'origine d'un client c qu'il ne contient pas, et le faire monter. Si le taxi arrive avant l'heure idéale de départ de c, il peut aller chercher une place de stationnement pour l'attendre. Lorsque c monte, la capacité du taxi ne peut pas être excédée.

Dans notre modèle, les contraintes liées au partage de courses sont de trois types. Premièrement, certains clients n'acceptent aucun partage de course. Deuxièmement, le nombre de places des taxis doit être respecté. Troisièmement, les contraintes de temps des clients indiquées dans la figure 3.4 doivent être respectées.

Lorsqu'un client est récupéré par un taxi, sa seule garantie est qu'il sera déposé avant son heure maximale d'arrivée, et que le taxi empruntera le plus court chemin pour tout déplacement. La course de la figure 3.5 est donc valide.

A chaque fois qu'un taxi s'arrête pour récupérer ou déposer un client, il doit attendre une durée fixe que le client monte ou descende, indépendamment du nombre de passagers du client. Si un taxi doit déposer un client dans une rue, puis récupérer un autre client dans cette même rue, on compte deux arrêts distincts.

3.2 Activités

Ayant défini les entrées du problème (taxis, chargeurs, requêtes, contraintes), on peut maintenant définir ce qu'est une solution et une solution optimale.

On s'intéresse à ce que font les taxis exclusivement durant l'intervalle de temps entre l'instant t_0 et l'instant t_{fin}, qui commence suffisamment tôt pour que chaque taxi soit capable de traiter n'importe quelle requête, et qui finit suffisamment tard pour que la dernière requête puisse être traitée, même avec le plus grand retard.

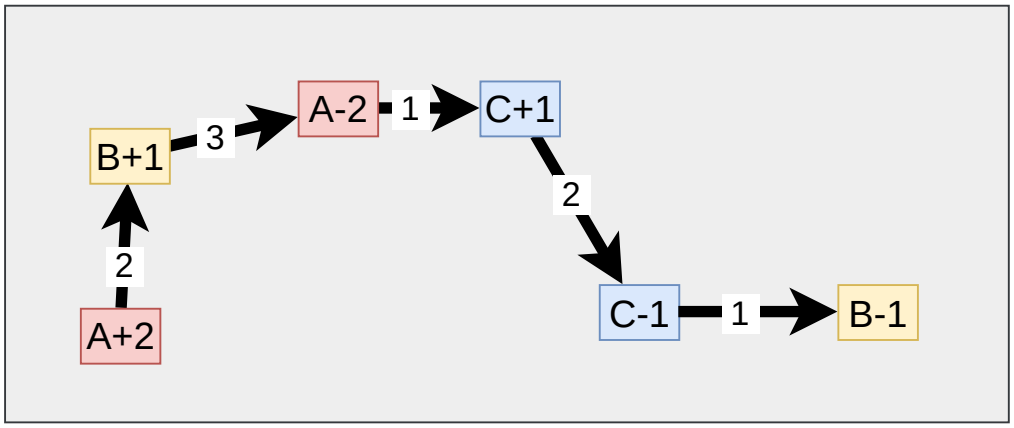


Figure 3.5 – *Une course partagée. "A+2" représente la montée du client A, qui compte 2 passagers, et "A-2" est sa descente. Le nombre sur chaque arc indique le nombre de passagers dans le taxi.*

Dans une solution, chaque taxi accomplit des actions. Ces actions peuvent être regroupées en activités, ces activités peuvent être regroupées en un emploi du temps de taxi, et l'ensemble des emplois du temps de taxis forme une solution.

Comme on l'a vu dans le chapitre 2, le graphe du E-DARP est tel que chaque sommet représente une récupération de client ou une descente de client, et une solution est un chemin dans ce graphe représentant l'emploi du temps d'un taxi, et qui respecte un ensemble de conditions.

Il est possible de modéliser le PCE-ADARP de la même manière. Cela étant, afin de faciliter la compréhension des heuristiques des chapitres 6 et 7, et afin de rendre plus intéressante l'analyse du comportement de ces méthodes du chapitre 9, on définit le modèle à l'aide des concepts d'action et d'activité présentés dans la présente section.

3.2.1 Actions

Voici les différentes actions que les taxis peuvent effectuer :

1. Rechercher une place de stationnement, et attente une fois cette place trouvée.
2. Manœuvrer pour se connecter à un chargeur, puis se charger.
3. Se déplacer, hors recherche de place et manœuvres.
4. Récupérer ou déposer un client (Gestion de stop). Cette action commence avec une ouverture de porte du taxi et finit avec une fermeture de porte. On parle aussi de *montée de client* et de *descente de client*.

Chaque action est caractérisée par :

— Un type, parmi les quatre susmentionnés : stationnement (1), charge (2), déplacement (3), et gestion de stop (4).

— Un plus court chemin dans le graphe de la ville. Pour une action non-déplacement, ce chemin est composé d'un unique sommet du graphe de la ville.

— Une requête pour une gestion de stop.

— Un chargeur pour une action de charge.

Une action n'a donc pas de durée, d'instant de début et n'est associée à aucun taxi en particulier.

3.2.2 Activités

Une action n'est qu'un potentiel. Une activité est une action concrétisée (réalisée), ou une suite d'actions concrétisées.

Une activité est caractérisée par :

— Un taxi

— Un instant de début

— Un niveau de batterie du taxi à l'instant de début

— Une suite d'actions, telle que chaque action commence dans le sommet du graphe de la ville où la précédente finit.

— Une fonction attribuant une durée à chaque action. Les actions de stop ont une durée prédéterminée, chaque action de déplacement a une durée déterminée par le trajet effectué et l'heure de départ (qui détermine la congestion), et les actions de charge et de stationnement ont une durée libre.

— Un type.

On distingue 4 types d'activités :

Trajet à vide : Contient une unique action de déplacement alors que le taxi est sans passager.

Stationnement : Contient une unique action de stationnement.

Charge : Contient une unique action de charge.

Course : Voir ci-dessous.

Une course contient une suite d'actions qui commence avec l'entrée d'un client dans un taxi vide, *i.e.* une gestion de stop. Une course se termine dès que le taxi redevient vide. Une course alterne donc déplacement (vers un stop) et gestion de stop. Le taxi peut arriver à un stop avant le client qu'il compte y récupérer. Dans ce cas, l'action de stop est précédée par l'action de stationnement. Par simplicité, on suppose qu'une action de gestion de stop après la première est toujours précédée d'une action de déplacement et d'une action de stationnement, éventuellement de durée nulle. La figure 3.6 présente visuellement la structure de toute course et la figure 3.7 présente un exemple de course partagée.

Une *portion de course* désigne l'ensemble des actions entre la fin d'une gestion de stop (ou le début de la course) et la fin de la gestion de stop suivante.

La durée des actions de stationnement est contrainte, de sorte que l'action de récupération de chaque client a lieu entre son instant idéal de départ et son instant de départ au plus tard. Le taxi ne peut évidemment jamais contenir plus de passagers qu'il n'a de places.

Les activités ont d'autres caractéristiques, déductibles à partir des caractéristiques déjà présentées, à savoir :

— Une origine (l'origine de sa première action)

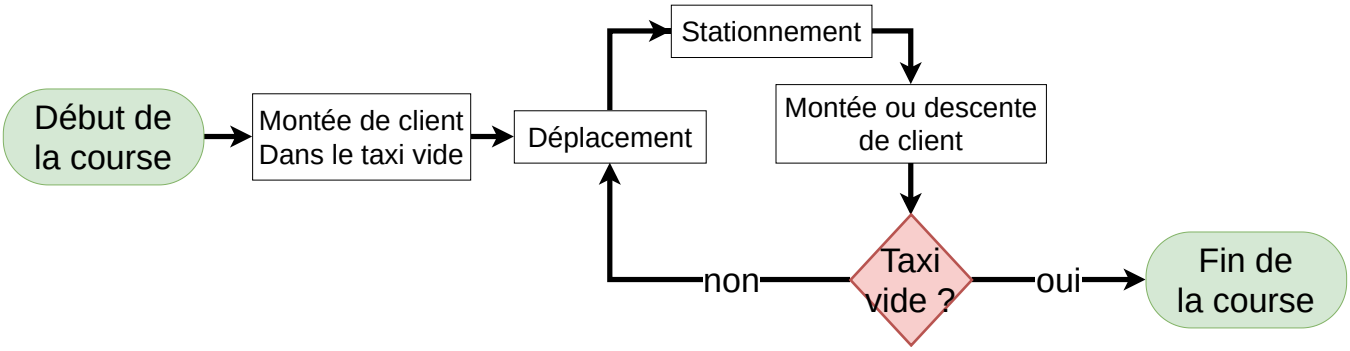


Figure 3.6 – *Toute course a une structure compatible avec ce diagramme. Le nombre d'actions dans une course est donc $6Q - 2$, avec Q le nombre de requêtes traitées par cette course.*

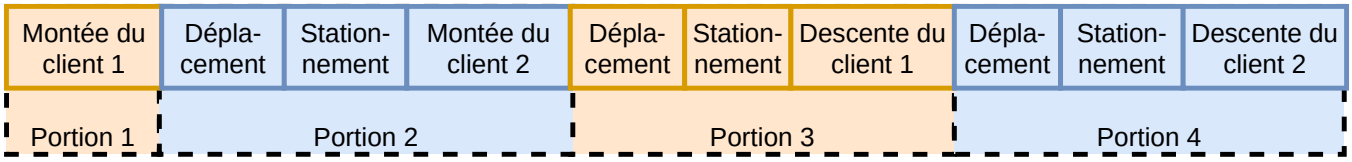


Figure 3.7 – *Example de suite des actions dans une course partagée avec deux clients. Cette suite peut être découpée en portions entre chaque stop. La première portion ne contient qu'une action, mais on peut supposer pour simplifier qu'elle contient aussi un déplacement et un stationnement, chacun de durée nulle.*

— Une destination (la destination de sa dernière action).

— Un instant de fin

— Une variation énergétique, somme de la variation énergétique de chaque action. La variation énergétique de chaque action est fonction de son type, ainsi que des caractéristiques du taxi et de la durée attribuée à cette action dans l'activité.

3.3 Solution et objectif

3.3.1 Emploi du Temps de Taxi (EDTT) et de Chargeur (EDTC)

Ayant établi ce que les taxis peuvent faire, et l'environnement dans lequel ils évoluent, on peut définir ce qu'est un emploi du temps d'un taxi (edtt) et un emploi du temps d'un chargeur (edtc).

Un edtt est une séquence contiguë d'activités, *i.e.* une séquence d'activités telle que chaque activité commence à l'endroit et l'instant où la précédente se termine, et telle que le niveau de batterie du taxi au début d'une activité correspond au niveau de batterie à la fin de la précédente. Le niveau de batterie d'un taxi ne peut évidemment pas devenir négatif.

La première activité est une phase de stationnement qui commence à l'instant t_0 à l'emplacement initial du taxi, et la dernière est aussi une phase de stationnement, qui finit à l'instant t_{fin} dans n'importe quel sommet du graphe de la ville. La figure 3.8 présente un tel edtt.

Le niveau de charge du taxi au début de la première activité est égal au niveau de batterie initial du taxi. Le niveau de charge après la dernière activité doit être supérieur au niveau

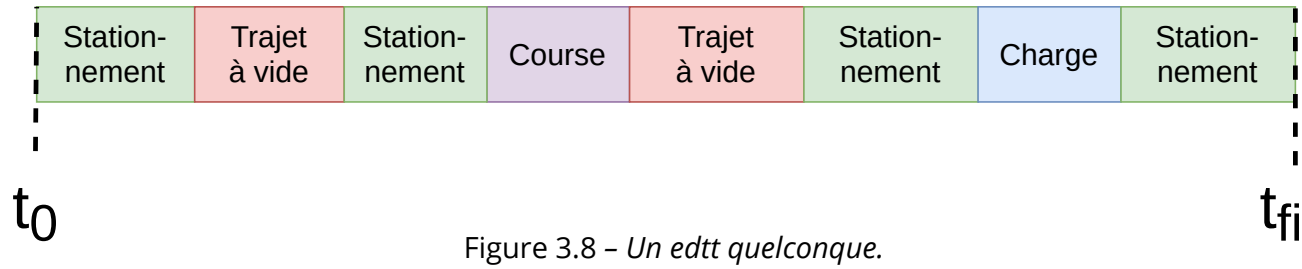


Figure 3.8 – *Un edtt quelconque.*

final minimal de batterie.

Un edtc est une séquence de charges à un même chargeur, telle que chaque charge commence après la fin de la précédente (il peut y avoir une période d'inactivité entre les deux).

3.3.2 Solution et fonction objectif

Une solution consiste en l'attribution d'un edtt pour chaque taxi. Ces edtt doivent être compatibles entre eux, *i.e.* un client ne peut pas être traité dans deux edtt. Les edtc peuvent être déduits à partir de l'ensemble des edtt. Dans la suite de ce mémoire, ce sont principalement les edtt qui sont manipulés, plutôt que les edtc.

La valeur d'une solution est la somme des valeurs des requêtes traitées par cette solution. La valeur d'une requête est, pour rappel, le produit de son nombre de passagers et de sa durée prévue. Une solution est optimale si et seulement si elle est de valeur maximale.

Réflexions sur l'objectif

Dans cette sous-section, la pertinence de la fonction objectif présentée ci-dessus est justifiée.

Dans le PCE-ADARP, trois choses doivent être optimisées. L'utilité du service (la qualité de service) doit être maximisée, son efficacité économique doit être maximisée, et sa consommation énergétique doit être minimisée.

La performance économique est un critère difficile à évaluer, dans la mesure où les services de partage de courses dans lesquels les clients ont une origine et destination différentes n'a pas encore vu émerger un modèle de tarification dominant. De plus, la tarification des services de véhicules autonomes sera vraisemblablement différente de celles des taxis traditionnels à laquelle on pourrait la comparer. Qui plus est, une solution à la fois bonne du point de vue de la qualité de service et de la consommation énergétique est sans doute économiquement bonne, donc ce critère est redondant avec les autres.

Pour juger de l'utilité d'une solution, l'indicateur de base choisi est une variante du voyageur-kilomètre. Dans le domaine du transport de personnes, le voyageur-kilomètre est couramment utilisé comme unité de mesure pour quantifier un volume de transport [website, 8]. L'INSEE le définit comme "une unité de mesure qui équivaut au transport d'un voyageur sur une distance d'un kilomètre". L'organisation de l'aviation civile internationale parle de "passager-kilomètre payant".

A taille de flotte de véhicules égale, dès lors que le calcul des bénéfices économiques est peu pratique, cette mesure semble pertinente pour juger de l'efficacité d'un service. Elle est

cependant insuffisante en elle-même pour ce but ; elle requiert plusieurs ajustements pour être adaptée à un service de taxis électriques autonomes partagés.

Premièrement, le partage de course implique des détours, et donc des kilomètres parcourus subis par les passagers. Maximiser le nombre de voyageur-kilomètres revient donc à essayer de faire le maximum de détours pour traiter les clients. Pour cette raison, il paraît pertinent de remplacer les voyageurs-kilomètres par une nouvelle métrique, les voyageurs-kilomètres utiles, qui ignorent ces détours. Dans notre modèle, la valeur d'une requête (définie dans la section 3.1.4) est son nombre de voyageurs-kilomètres utiles.

La qualité de service des clients traités est déterminée par le retard qu'ils subissent et la probabilité qu'une requête soit rejetée. Dans notre modèle, des contraintes de temps strictes sont utilisées (contrairement à certains DARP dans lesquels le retard toléré est infini), donc il n'est pas nécessaire d'intégrer le retard dans la fonction objectif.

Enfin, concernant la consommation énergétique, il est clair qu'une gestion inefficace de la recharge a pour conséquence de réduire le nombre de voyageurs-kilomètres. Il n'est donc pas non plus nécessaire de le prendre explicitement en compte dans l'objectif.

Le retard et la consommation énergétique auraient malgré tout pu être pris en compte dans la fonction objectif. La principale raison qui a amené à ne pas le faire est la difficulté de pondérer leur importance relative par rapport aux autres critères. Une autre raison est qu'une fonction objectif simple permet de rendre le modèle plus universel, et plus comparable.

Ce sont ces considérations qui nous ont amené à une fonction objectif dans laquelle on cherche à maximiser la valeur totale des requêtes, la valeur d'une requête étant proportionnelle à la fois à son nombre de passagers et à la durée du plus court chemin entre son origine et sa destination.

3.3.3 Génération de solution pour le problème dynamique

Dans le problème dynamique, la génération d'une solution est contrainte. La solution est générée incrémentalement, à l'aide d'une variable qu'on appelle $present$ et qui ne peut que croître. La partie d'un edtt précédant l'instant $present$ ne peut pas être modifié. Soit S une solution. Pour que S soit transformée en une solution S', les règles suivantes doivent être respectées.

Toute activité de S finissant avant $present$ est aussi dans S'. Concernant les activités commençant avant $present$ et finissant après, elles peuvent être interrompues, mais en respectant des contraintes liées à leur type.

Stationnement Une phase de stationnement peut être rallongée ou raccourcie, tant qu'elle continue de finir après $present$.

Charge Une charge a les mêmes contraintes qu'une phase de stationnement, mais sa phase de manœuvre ne peut pas être raccourcie. En d'autres termes, une charge ne peut pas avoir une durée inférieure à a (voir figure 3.3). A noter que, dans la version statique du problème, les charges d'une durée inférieure à b ou supérieure à c peuvent être interdites, car n'ayant pas d'intérêt.

Trajet à vide Aucune modification n'est tolérée, le taxi doit atteindre sa destination.

Course Si, à l'instant $present$, le taxi est entre l'arrêt i et l'arrêt $i+1$ de sa course, son edtt ne peut pas être modifié avant qu'il n'ait atteint l'arrêt $i+1$.

Enfin, toute requête traitée dans S est traitée dans S', et une unique requête dont l'heure de formulation est $present$ peut être traitée dans S' sans être traitée dans S.

3.4 Graphe des actions

A ce stade de ce chapitre, l'ensemble des entrées, sorties et contraintes du PCE-ADARP ont été présentées. Dans le chapitre suivant, elles sont modélisées formellement et mathématiquement. Afin de simplifier cette modélisation, un nouveau graphe est utilisé, qu'on appelle *graphe des actions*.

Le graphe des actions représente toutes les actions que les taxis sont susceptibles de faire. Il est généré à partir du graphe de la ville, et de l'ensemble des taxis, chargeurs et requêtes.

Ce graphe correspond à ce qu'on appelle parfois dans la littérature sur le VRP le *Customer-Based Graph* (graphe des clients), notamment utilisé par [Ben Ticha et al., 2018]. Dans le modèle mathématique du chapitre 4, c'est surtout le graphe des actions qui est manipulé.

3.4.1 Structure

Dans le graphe des actions, chaque sommet représente des actions associées au même sommet du graphe de la ville. L'ensemble des sommets du graphe des actions peut être divisé en 3 sous-ensembles, qui sont D, C et S.

Soit v un sommet du graphe des actions.

— Si $v \in D$, alors v représente une action de stationnement.

— Si $v \in C$, alors v représente une action de stationnement suivie d'une action de **c**harge à un chargeur donné, suivie d'une action de stationnement.

— Si $v \in S$, alors v représente une action de stationnement suivie d'une gestion de **s**top. L'ensemble S peut être divisé en deux.

- S^+ est l'ensemble des montées de clients.
- S^- est l'ensemble des descentes de clients.

Détaillons les caractéristiques de chaque ensemble de sommets.

Sommets de début (D)

Chaque taxi x est associé à un unique sommet de D. L'action de stationnement du sommet de D est associée au point de départ du taxi dans le graphe de la ville.

Sommets de gestion de stop (S)

Pour chaque client/requête q, il y a un unique sommet $s_q^+ \in S^+$ et un unique sommet $s_q^- \in S^-$. Le sommet s_q^+ représente la récupération du client de q, et les trois actions associées à ce sommet sont associées à l'origine de q dans le graphe de la ville. De la même manière, le sommet s_q^- est associé à la destination de q.

Sommets de charge (C)

Pour chaque chargeur $\mathfrak{c}$, on a $C^{\mathfrak{c}} \subseteq C$. Les actions associées à ce sommet sont associées au sommet du graphe de la ville lui-même associé à $\mathfrak{c}$. On a $|C^{\mathfrak{c}}| = t_{fin} - t_0$.

Arcs

Chaque arc (u, v) du graphe des actions représente une action de déplacement, dont l'origine est le sommet du graphe de la ville associé à u et la destination est le sommet associé à v. La figure 3.9 présente l'ensemble des arcs présents dans le graphe des actions.

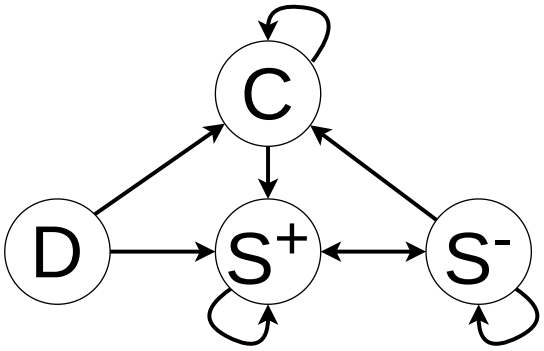


Figure 3.9 – *Représentation simplifiée du graphe des actions. Chaque sommet de ce graphe représente un des ensembles de sommets du graphe des actions. Un arc (A, B) dans cette image signifie qu'il existe un arc depuis chaque sommet de A vers chaque sommet de B, avec 3 exceptions. Il n'y a pas de boucle (arc (v, v) avec $v \in V$). Il n'y a pas d'arc (s_q^-, s_q^+) avec q une requête. La dernière exception est détaillée dans la sous-section "Solution". Un emploi du temps dans lequel un taxi traite un unique client puis se charge peut être représenté par un chemin $D \to S^+ \to S^- \to C$*

Le graphe de la figure 3.10 présente un exemple de graphe des actions dans une ville. La section suivante présente la relation entre le graphe des actions et une solution du problème.

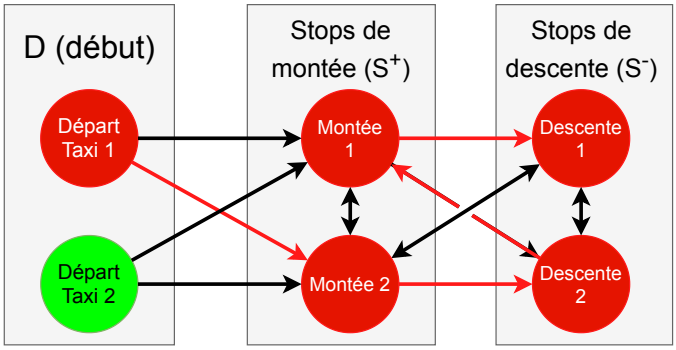


Figure 3.10 – *Graphe des actions, dans une ville avec 2 taxis, 2 requêtes, et 0 chargeur. Ce graphe est équivalent à un Customer-Based Graph d'un PDP. Les chemins rouge et vert représentent respectivement les edtt d'une solution (voir fin de la section 3.4.2). Le chemin vert correpond à un edtt dans lequel le taxi reste stationné de t_0 à t_{fin}.*

3.4.2 Solution

Une solution est un ensemble d'edtt (*cf.* section 3.3.1) qui sont des suites d'activités (*cf.* section 3.2.2) qui sont des suites d'actions (*cf.* section 3.2.1).

Définissons une solution par rapport au graphe des actions.

Activités

Une activité est une suite d'actions associée à un ensemble de variables définies dans la section 3.2.2.

Cette suite d'actions peut être représentée par un chemin dans le graphe des actions (une suite de sommets). Soit Z le chemin représentant une activité act. Voici sa forme, en fonction du type de l'activité.

Si act est une course $Z = (s_1, ..., s_n) \subseteq S$ avec $s_1 \in S^+, s_n \in S^-$
Si act est un trajet à vide $Z = (u, v)$ avec $u \notin S^+$ et $v \in V$.
Si act est une activité de charge $Z = (c)$ avec $c \in C$.
Si act est une activité de stationnement $Z = (v)$ avec $v \in V$.

Une portion de course (*cf.* figure 3.6) est soit une gestion de stop, s'il s'agit de la première portion d'une course, soit une suite de 3 actions dont les types sont, dans l'ordre, déplacement, stationnement, et gestion de stop.

La première portion d'une course est représentée avec un unique sommet de S^+. Chacune des autres portions est représentée par un arc dont la destination est $s \in S$ suivi de s. Si $s \in S^-$, l'action de stationnement de la portion est de durée nulle. Sinon, l'action de stationnement a une durée non nulle si le taxi arrive avant le client au stop.

Pour toute requête q, une course ne peut pas passer par le sommet s_q^- si le sommet s_q^+ n'a pas déjà été parcouru durant la course.

Les variables associées à une activité (durée, batterie initiale, ...) ne peuvent pas être déterminées à l'aide du graphe des actions, avec deux exceptions. Soit $d \in D$, qui est associé au taxi x. Une activité de stationnement associée à d doit avoir pour taxi x, pour instant de début t_0, et pour batterie initiale la batterie initiale de x.

Edtt

Un edtt est une suite d'activités respectant certaines contraintes mentionnées dans la section 3.3.1.

La première activité d'un edtt e associé à un taxi x est une activité de stationnement dans le sommet de D associé à x. La dernière activité de e est un stationnement finissant à l'instant t_{fin}.

Un edtt est contigu, ce qui signifie entre autres que le chemin du graphe des actions représentant une activité doit commencer là où celui de la précédente finit.

Les figures 3.11 et 3.12 montrent la relation entre un edtt et un chemin dans le graphe des actions le représentant.

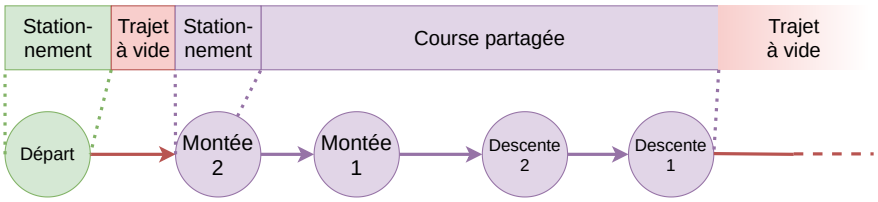


Figure 3.11 – *Un chemin dans le graphe des actions (en bas) est découpé en cinq activités (en haut) qui forment un début d'edtt. En l'occurrence, il s'agit du début de l'edtt du taxi 1 de la figure 3.10 ou de la figure 3.13.*

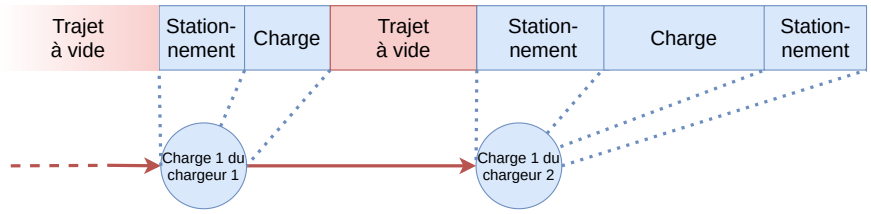


Figure 3.12 – *Un chemin dans le graphe des actions (en bas) est découpé en sept activités (en haut) qui forment une fin d'edtt. En l'occurrence, il s'agit de la fin de l'edtt du taxi 1 de la figure 3.13*

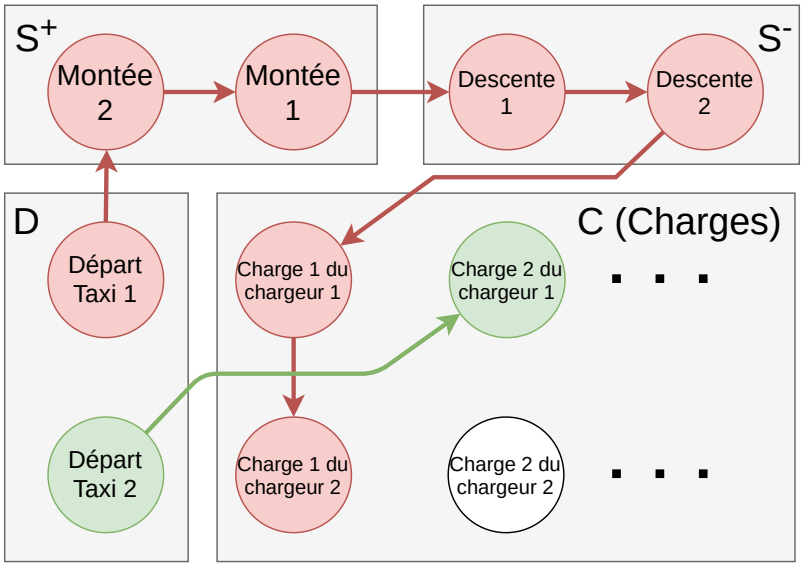


Figure 3.13 – *Un graphe des actions et une solution. les chemins rouge et vert représentent respectivement l'edtt des taxis 1 et 2 dans une solution. Pour une question de visibilité, seuls les arcs faisant partie de la solution sont montrés ici.*

Solution

Une solution pour le PCE-ADARP étant un ensemble d'edtt, et puisque un client ne peut être traité qu'une seule fois, les chemins formés par les edtt dans une solution sont sommet-disjoints. La figure 3.13 présente le chemin formé par une solution dans le graphe des actions.

Comme il a été mentionné dans la section 3.3.1, un edtc (edtc = suite de charges à un même chargeur) peut être généré à partir des edtt. Dans un edtc, chaque charge commence après la fin de la précédente, avec éventuellement une période d'inactivité entre les deux. Dans le graphe des actions, on modélise cette contrainte de la manière suivante.

Chaque sommet de C a un numéro attribué. Soit c_i et c_j deux sommets de C associés à un chargeur $\mathfrak{c}$, tel que $i < j$. Intuitivement, c_i représente la i-ème charge au chargeur $\mathfrak{c}$. Si c_i n'est associé à aucune activité de charge, alors c_j n'est pas non plus associé à une activité de charge. Si c_j est associé à une activité de charge, alors c_i l'est aussi, et la charge de c_j commence après que celle de c_i finisse. Il n'y a pas d'arc (c_j, c_i).

Enfin, dans le PCE-ADARP dynamique, et/ou lorsque la congestion est variable, une activité de course ou de charge peut être suivie d'une activité de stationnement.

3.4.3 Conclusion

Dans ce chapitre, le problème pratique étudié dans cette thèse a d'abord été présenté, ainsi que les hypothèses sous-jacentes qui le caractérisent. La relation entre ce problème pratique et le PCE-ADARP, qui est un problème mathématique, a également été présentée, et nos partis-pris de modélisation ont été justifiés. Ce travail ayant été effectué, on peut maintenant présenter en détails le modèle mathématique.

Chapitre 4

Modélisation du problème

Dans ce chapitre, une description mathématique du problème est présentée. On commence avec la description des données du problème statique, puis d'une solution pour ce problème, puis enfin les particularités du problème dynamique.

Dans la littérature, le VRP et ses variantes sont plutôt présentés sous la forme de système d'inéquation linéaires. Une telle formulation est présentée dans l'annexe A.

La formulation du présent chapitre est plus adaptée pour comprendre les chapitres 6, 7 et 9.

Notations

Voici un ensemble de notations qui seront utilisées dans la suite de ce mémoire.

Soit Ens un ensemble quelconque. La notation $\langle Ens \rangle$ sert à indiquer que cet ensemble est ordonné et fini, et tel que ens_n est le n-ième élément, en partant du début si n est positif, ou le n-ième élément en partant de la fin si n est négatif. Par exemple, ens_1 et ens_{-1} dénotent le premier et dernier élément de $\langle Ens \rangle$, respectivement.

On note $\mathbb{N}_{[a,b]}$ l'ensemble des entiers dans l'intervalle $[a, b]$.

Le modèle est en temps discret. Chaque instant est désigné par un entier. Soit t un intervalle de temps. La notation $t = [a, b]$ signifie que $t = \mathbb{N}_{[a,b]}$. On note aussi $\alpha(t) = a$ son début et $\omega(t) = b$ sa fin.

4.1 Données du problème

Les quatre principales entrées du problème sont : le graphe de la ville, les taxis, les requêtes et les chargeurs.
Dans cette section, les variables composant ces entrées sont listées exhaustivement.

Les taxis peuvent bouger dans l'intervalle de temps $T = [t_0, t_{fin}]$.

4.1.1 Structure du graphe de la ville

Une description de ce que représente le graphe de la ville se trouve à la page 29.
Ce graphe est nommé $G^{ville} = (V^{ville}, A^{ville})$. Il est connexe et orienté. Chaque arc $a \in A^{ville}$ a une longueur $l_a \in \mathbb{R}^+$ et une vitesse $vitesse_a \in \mathbb{R}^+$.

La ville a une valeur de congestion à chaque instant $t \in T$ qu'on note $congestion_t \in \mathbb{R}^+$.

On note $cherche_{max} \in \mathbb{N}^+$ la durée maximale d'une recherche de place de stationnement, dans n'importe quel sommet.

4.1.2 Composants d'un taxi

Une description des taxis se trouve à la page 31. L'ensemble des taxis est appelé X, tel que chaque taxi $x \in X$ a :

- Un nombre de places $places_x \in \mathbb{N}^+$
- Un point de départ d_x^{ville} dans V^{ville}.
- Une consommation énergétique par instant $\Delta_x^{temps} \in \mathbb{R}^+$
- Une consommation énergétique par mètre parcouru $\Delta_x^{metre} \in \mathbb{R}^+$
- Une quantité maximale d'énergie dans la batterie $w_x^{max} \in \mathbb{R}^+$
- Une quantité d'énergie $w_x^0 \in \mathbb{R}^+$ à l'instant t_0.

Un taxi va à une vitesse constante $croisiere \in \mathbb{R}^+$ lors d'une phase de recherche de stationnement.

La variable $vitesse\_mult \in \mathbb{R}$ influence la consommation par mètre des taxis en fonction de leur vitesse.

Les taxis ont une proportion d'énergie finale $w^{fin} \in \mathbb{R}_{[0,1]}$.

4.1.3 Composants d'une requête

Une description des requêtes se trouve à la page 32. L'ensemble des requêtes est Q, tel que $\forall\, q \in Q$, on a :

- Un nombre de passagers $passagers_q \in \mathbb{N}^+$
- Un sommet de montée $s_q^{+,ville} \in V^{ville}$
- Un sommet de descente $s_q^{-,ville} \in V^{ville}$
- Une fenêtre de temps $montee_q \subseteq T$ (instant de départ au plus tôt $\rightarrow$ instant de départ au plus tard)
- Une fenêtre de temps $descente_q \subseteq T$ (instant d'arrivée au plus tôt $\rightarrow$ instant d'arrivée au plus tard).

On note $duree\_stop \in \mathbb{N}^+$ la durée nécessaire pour récupérer ou déposer un client.

4.1.4 Composants d'un chargeur

Une description des chargeurs se trouve à la page 32. L'ensemble des chargeurs est $\mathfrak{C}$. Chaque chargeur $\mathfrak{c} \in \mathfrak{C}$ a une puissance $w_{\mathfrak{c}} \in \mathbb{R}^+$ et un emplacement $v_{\mathfrak{c}}^{ville} \in V^{ville}$.

A chaque instant, chaque chargeur est utilisable ou inutilisable. Une hypothèse simplificatrice que l'on émet est que, à chaque instant, soit tous les chargeurs sont utilisables, soit aucun ne l'est.
L'ensemble des intervalles de temps durant lesquels les chargeurs sont utilisables/disponibles est $\langle Dispo \rangle$, trié par ordre chronologique, *i.e.* tel que $\omega(dispo_i) \leq \alpha(dispo_{i+1})$.

On note $manoeuvre \in \mathbb{N}^+$ la durée d'une manœuvre pour se connecter à un chargeur.

4.1.5 Instance du problème

Une *instance* est un ensemble contenant toutes ces données d'entrée. Autrement dit, une instance I est un ensemble tel que :

$$I = \{G^{ville}, Q, X, \mathfrak{C}, cherche_{max}, manoeuvre, duree\_stop, croisiere, \\ w^{fin}, t_0, t_{fin}, \{congestion_t \mid \forall\, t \in T\}, vitesse\_mult, \langle Dispo \rangle\}$$

4.2 Solution du problème

Une solution est définie à partir du graphe des actions présenté dans la section 3.4. C'est un ensemble d'edtt (*cf.* section 3.3.1) qui sont des suites d'activités (*cf.* section 3.2.2) qui sont des suites d'actions (*cf.* section 3.2.1).

4.2.1 Structure du graphe des actions

Le graphe $G = (V, A)$ est le graphe des actions. Récapitulons ses caractéristiques. On a $V = \{D \cup S \cup C\}$ avec D, S et C disjoints, tels que :

- D est l'ensemble des actions de stationnement initial des taxis
- S est l'ensemble des actions de gestions de stop (en abrégé, l'ensemble des stops). Il est divisé en deux :
  - S^+ est l'ensemble des montées de client
  - S^- est l'ensemble des descentes de client
- C est l'ensemble des actions de charge

On a $|D| = |X|$ et $|S^+| = |S^-| = |Q|$.
On note $ville(v)$ le sommet du graphe de la ville associé au sommet $v \in V$.
A chaque taxi $x \in X$ est associé un sommet $d_x \in D$ tel que $d_x^{ville} = ville(d_x)$.

A chaque requête $q \in Q$ est associée un sommet $s_q^+ \in S^+$ et un sommet $s_q^- \in S^-$ tels que $ville(s_q^+) = s_q^{+,ville}$ et $ville(s_q^-) = s_q^{-,ville}$. On note $[\alpha_v, \omega_v] \subseteq T$ la fenêtre de temps du sommet $v \in S$. Cette fenêtre de temps est telle que $\exists!\ q \in Q$:

$$v = s_q^+ \Rightarrow \begin{cases} \alpha_v = \alpha(montee_q) \\ \omega_v = \omega(montee_q) \end{cases}$$

ou

$$v = s_q^- \Rightarrow \begin{cases} \alpha_v = \alpha(descente_q) \\ \omega_v = \omega(descente_q) \end{cases}$$

A chaque chargeur $\mathfrak{c} \in \mathfrak{C}$ est associé une suite $\langle C^{\mathfrak{c}} \rangle \subseteq C$ telle que $|C^{\mathfrak{c}}| = t_{fin} - t_0$. Chaque sommet représente une charge potentielle (il y a au plus une charge commençant à chaque instant à un chargeur donné).

L'ensemble des arcs du graphe des actions est présenté dans la figure 3.9. Chaque arc $a \in A$ a une longueur $l_a \in \mathbb{R}^+$ et une durée de base $\tau(a) \in \mathbb{R}_{[0,+\infty[}$. Avec $(u, v) \in A$, on note $ville((u, v))$ la suite des arcs (chemin) de plus petite durée totale de $ville(u)$ vers $ville(v)$, en ignorant la congestion.

La longueur de l'arc (u, v) du graphe des actions n'est pas tout à fait la somme des longueurs des arcs de $ville((u, v))$. Puisque la longueur de (u, v) n'est utilisée que dans des calculs de consommation énergétique, et pour accélérer les calculs, on donne à l'arc (u, v) la longueur d'un trajet consommant autant d'énergie et effectué à vitesse de croisière. On a donc, $\forall\ a \in A$:

$$l_a = \sum_{rue \in ville(a)} l_{rue} \times (1 + vitesse\_mult \times |croisiere - vitesse_{rue}|) \tag{4.1}$$

$$\tau(a) = \sum_{rue \in ville(a)} \frac{l_{rue}}{vitesse_{rue}} \tag{4.2}$$

Le paramètre $vitesse\_mult$ n'est utilisée nulle part ailleurs.

4.2.2 Composition des actions

Soit a une action (*cf.* page 37).
a est définie par un type qu'on note $type(a)$, qui est soit stationnement, soit charge, soit déplacement, soit stop.

Un stationnement est associé à chaque sommet de V.
Un déplacement d est associée à chaque arc de A. On note $a(d) \in A$ l'arc associé à d.
Une charge est associée à chaque sommet de C.
Une gestion de stop est associée à chaque sommet de S. On note $v(act) \in V$ le sommet associé à l'action act qui n'est pas un déplacement.
On note $\mathfrak{c}(c)$ le chargeur associé à l'action de charge c tel que $v(c) \in C^{\mathfrak{c}}$.

On note $\Delta_{croisiere}(x,t)$ la variation énergétique d'un taxi $x \in X$ qui se déplace à une vitesse de croisière pendant une durée $t \in \mathbb{N}^+$. Cette fonction est telle que :

$$\Delta_{croisiere}(x,t) = -t \times (\Delta_x^{temps} + croisiere \times \Delta_x^{metre}) \tag{4.3}$$

On note $\Delta(act,x,t) \in \mathbb{R}$ la variation énergétique d'un taxi x effectuant l'action act pendant une durée t.

$$\begin{aligned}
\Delta(act,x,t) &= \Delta_{croisiere}(x, min(t, cherche_{max})) && \text{si } type(act) = \text{stationnement} && (4.4)\\
&= -l_{a(act)} \times \Delta_x^{metre} - t \times \Delta_x^{temps} && \text{si } type(act) = \text{déplacement} && (4.5)\\
&= \Delta_{croisiere}(x, duree_stop) && \text{si } type(act) = \text{stop} && (4.6)\\
&= (t - manoeuvre) \times w(\mathfrak{t}(c)) \\
&\quad + \Delta_{croisiere}(x, manoeuvre) && \text{si } type(act) = \text{charge} && (4.7)
\end{aligned}$$

4.2.3 Activité

Une activité a (*cf.* page 38) est composée de quatre éléments. Un taxi $x(a) \in X$, un instant de début $debut(a) \in T$, une suite d'actions $\langle Action^a \rangle$, et une fonction attribuant une durée à chaque action. Ces éléments peuvent être utilisés pour associer d'autres valeurs à cette activité :

- $fin(a) \in T$
- $origine(a) \in V$
- $destination(a) \in V$
- $duree(a) = fin(a) - debut(a)$
- $\Delta(a) \in \mathbb{R}$ sa variation énergétique.

Les activités sont de quatre types, et chaque type a des contraintes et des variables qui lui sont propre. Ces types sont : stationnement, trajet-à-vide, charge, et course. On note $type(a)$ le type de l'activité a.

Stationnement

Soit act une activité de stationnement. On dit que act est valide si et seulement si, avec $s = action_1^{act}$, on a $s = action_{-1}^{act}$ et $type(s) =$ stationnement. On note :

$$origine(act) = destination(act) = v(s) \tag{4.8}$$
$$fin(act) \geq debut(act) \tag{4.9}$$
$$\Delta(act) = \Delta(s, x(act), min(duree(s), cherche_{max})) \tag{4.10}$$

Trajet à vide

Soit act un trajet à vide. On dit que act est valide si et seulement si, avec $d = action_1^{act}$, on a $d = action_{-1}^{act}$ et $type(d) =$ trajet-à-vide. On a donc, avec $a(d) = (u, v)$:

$$origine(act) = u \tag{4.11}$$
$$destination(act) = v \tag{4.12}$$
$$duree(act) = \tau(a(d)) \times congestion_{debut(act)} \tag{4.13}$$
$$\Delta(act) = \Delta(d, x(act), duree(act)) \tag{4.14}$$

Charge

Soit act une activité de charge. On dit que act est valide si et seulement si, avec $c = action_1^{act}$, on a $c = action_{-1}^{act}$ et $type(c) =$ charge. On a donc :

$$\exists\, d \in \langle Dispo \rangle, \qquad \alpha(d) \leq debut(c) \leq fin(c) \leq \omega(d) \tag{4.15}$$
$$duree(c) \geq manoeuvre \tag{4.16}$$

On appelle $\mathfrak{c}(act)$ le chargeur qui associé à act. On a :

$$origine(act) = destination(act) = v(c) \tag{4.17}$$
$$\Delta(c) = \Delta(c, x(act), duree(act)) \tag{4.18}$$
$$\mathfrak{c}(act) = \mathfrak{c}(c) \tag{4.19}$$

Portion de course

Avant de décrire les caractéristiques d'une course (voir section 3.2.2), décrivons une portion d'une course. La figure 3.7 présente une course divisée en portions. Une portion regroupe tout ce qui arrive entre deux fins de stop, et est donc une suite d'actions de la forme "déplacement, action de stationnement, gestion de stop".
Une portion p d'une course est définie par :

- $x(p) \in X$
- Une suite de 3 actions : p^1 un déplacement, p^2 un stationnement, et p^3 un stop
- $debut(p) \in T$
- $arrive(p) \in T$ (la fin de p^1)
- $fin(p) \in T$
- $\Delta(p) \in \mathbb{R}$

Pour que p soit valide, il faut qu'elle soit contiguë spatialement. notons $u, v \in S$ tel que $a(p^1) = (u, v)$:

$$v = v(p^2) = v(p^3) \tag{4.20}$$

temporellement :

$$arrive(p) = debut(p) + \tau(a(p^1)) \times congestion_{debut(p)} \tag{4.21}$$
$$fin(p) = max(arrive(p), \alpha_{v(p^2)}) + duree\_stop \tag{4.22}$$

énergétiquement :

$$\begin{aligned}\Delta(p) = &\Delta(p^1, x(p), \tau(a(p^1)) \times congestion_{debut(p)}) \\ &+ \Delta(p^2, x(p), fin(p) - duree\_stop - arrive(p)) \\ &+ \Delta(p^3, x(p), 0)\end{aligned} \tag{4.23}$$

et que les contraintes de temps du client soient respectées :

$$\alpha_{v(p^3)} \leq fin(p) \leq \omega_{v(p^3)} \tag{4.24}$$

La requête associée à la gestion de stops est noté $q(p)$ telle que $q(p) = q_{v(p^3)}$.
On note $v(p) = v(p^3)$. La variable booléenne $montee(p) \in \{0, 1\}$ indique si la gestion de stop correspond à une montée de client :

$$montee(p) = 1 \Leftrightarrow v(p) \in S^+ \tag{4.25}$$
$$montee(p) = 0 \Leftrightarrow v(p) \in S^- \tag{4.26}$$

Course

Une course c est composée d'une suite de portions de course valides qu'on note $\langle P^c \rangle$. Sa première action de déplacement et sa première action de stationnement sont de durée nulle.
Pour que la course c soit valide, elle doit être contiguë spatialement (4.27), temporellement (4.28) avec un taxi fixe (4.29). Donc, $\forall\ k \in \mathbb{N}^+$:

$$v(p_k^{c,1}) = v(p_{k-1}^{c,3}) \tag{4.27}$$
$$debut(p_k^c) = fin(p_{k-1}^c) \tag{4.28}$$
$$x(p_k^c) = x(c) \tag{4.29}$$

On note $passagers(k)$ le nombre de passagers dans le taxi au début de la portion p_k. On note $\Delta_{passagers}(k)$ la variation du nombre de passagers durant la portion p_k. Pour rappel, $passagers_q$ dénote le nombre de passagers de la requête q. On a donc :

$$passagers(1) = 0 \tag{4.30}$$
$$\forall\ k > 1,\ passagers(k) = passagers(k-1) + \Delta_{passagers}(k-1) \tag{4.31}$$
$$\forall\ k \in \mathbb{N}^+,\ \Delta_{passagers}(k) = \begin{cases} +passagers_{q(p_{k-1}^c)} \text{ si } montee(p_{k-1}^c) = 1 \\ -passagers_{q(p_{k-1}^c)} \text{ sinon} \end{cases} \tag{4.32}$$

Les contraintes de capacité du taxi doivent être respectées, et le taxi doit finir vide :

$$\forall\ k \in \mathbb{N}^+, 0 < passagers(k) \leq places_x \tag{4.33}$$
$$passagers(-1) + \Delta_{passagers}(-1) = 0 \tag{4.34}$$

Ayant défini les contraintes d'une course, on peut décrire ses caractéristiques :

$$origine(c) = v(p_1^{c,3}) \tag{4.35}$$
$$destination(c) = v(p_{-1}^{c,3}) \tag{4.36}$$
$$debut(c) = debut(p_1^c) \tag{4.37}$$
$$fin(c) = fin(p_{-1}^c) \tag{4.38}$$
$$\Delta(c) = \sum_{p \in \langle P^c \rangle} \Delta(p) \tag{4.39}$$

L'ensemble des requêtes traitées durant c est appelé
$Q(c) = \{q \in Q \mid \exists\, p \in \langle P^c \rangle$ tel que $q(p) = q\}$

4.2.4 Emploi du temps de taxi

Un edtt (*cf.* section 3.3.1) est une suite d'activités. Soit $\langle E \rangle$ un edtt quelconque. Notons $w(e) \in \mathbb{R}^+$ le niveau de batterie du taxi au début de l'activité e et $x(E)$ le taxi associé à $\langle E \rangle$. Pour que $\langle E \rangle$ soit valide, toutes ses activités doivent être associées à $x(E)$:

$$\forall\, e \in \langle E \rangle, x(e) = x(E) \tag{4.40}$$

Le taxi doit commencer dans le bon état :

$$v(e_1) = d_{x(E)} \tag{4.41}$$
$$debut(e_1) = t_0 \tag{4.42}$$
$$w(e_1) = w^0_{x(E)} \tag{4.43}$$

Le taxi doit finir dans le bon état. Pour rappel, w^{fin} est la proportion d'énergie requise dans la batterie des taxis à la fin de l'edtt :

$$fin(e_{-1}) = t_{fin} \tag{4.44}$$
$$w(e_{-1}) + \Delta(e_{-1}) \geq w_x^{max} \times w^{fin} \tag{4.45}$$

Cette suite d'activités doit être contiguë, autrement dit, $\forall\, k \in \mathbb{N}^+$:

$$fin(e_k) = debut(e_{k+1}) \tag{4.46}$$
$$destination(e_k) = origine(e_{k+1}) \tag{4.47}$$
$$w(e_k) = w(e_{k-1}) + \Delta(e_{k-1}) \tag{4.48}$$

Le niveau de batterie du taxi doit toujours être dans ses limites :

$$\forall\, k \in \mathbb{N}^+,\ 0 < w(e_k) \leq w^{max}_{x(E)} \tag{4.49}$$

L'ensemble des requêtes traitées durant $\langle E \rangle$ est noté :

$$Q(E) = \{q \in Q(c) \mid\ c \in E \text{ et } c \text{ est une course}\} \tag{4.50}$$

4.2.5 Solution

La notation $\langle E^x \rangle$ indique un edtt associé à un taxi x (tel que $x(E^x) = x$).
Une solution est un ensemble d'edtt $SOL = \{\langle E^x \rangle \forall\ x \in X\}$.
Ces edtt sont sommet-disjoints. $\forall\ E, E' \in SOL \mid\ E \neq E'$:

$$v(e) \in E \Rightarrow v(e) \notin E' \tag{4.51}$$

A partir de SOL, on peut générer les emplois du temps de chargeurs (edtc). Un edtc est un ensemble de charges ayant lieu au même chargeur (4.52), trié chronologiquement sans permettre de charges simultanées (4.53). Notons $\langle E^{\mathfrak{c}} \rangle$ l'edtc de $\mathfrak{c} \in \mathfrak{C}$. Voici ses caractéristiques :

$$\langle E^{\mathfrak{c}} \rangle = \langle e \in \bigcup_{\langle E \rangle \in SOL} \mid\ v(e) \in C^{\mathfrak{c}} \rangle \tag{4.52}$$

$$\forall\ k \in \mathbb{N}^+,\ debut(e^{\mathfrak{c}}_{k+1}) \geq fin(e^{\mathfrak{c}}_k) \tag{4.53}$$

L'ensemble des requêtes traitées dans la solution est $Q(SOL) = \{q \in Q(E) \mid\ E \in SOL\}$.
Chaque requête q a une valeur. En notant $d = \alpha_{s^+_q}$ l'instant de départ idéal du client, on a :

$$valeur(q) = passagers_q \times \tau((s^+_q, s^-_q)) \times congestion_d \tag{4.54}$$

L'objectif est de maximiser la valeur des requêtes traitées :

$$max \sum_{q \in Q(SOL)} valeur(q) \tag{4.55}$$

4.2.6 Particularités du problème dynamique

Dans le problème dynamique, chaque requête a un instant de formulation. Les requêtes sont ainsi découvertes progressivement, et la solution doit être adaptée à chaque découverte de requête, sans pouvoir modifier le passé ou retirer des requêtes de la solution. Le problème dynamique se distingue du problème statique de trois manières.

Premièrement, chaque requête a un instant de formulation inférieur à t_{fin}. Lorsqu'une requête est formulée, la solution actuelle doit être mise-à-jour pour traiter ce nouveau client, si cela est possible.

Deuxièmement, l'insertion des requêtes dans la solution est définitive. A partir du moment où une requête est acceptée, il n'est pas possible de changer la solution d'une manière qui fasse que la requête n'est plus traitée. Soit t l'instant de formulation de la requête $q \in Q$. Soit SOL la solution avant la formulation de q, et SOL' la solution après. Dans ce cas, $Q(SOL') = Q(SOL)$ ou $Q(SOL') = Q(SOL) \cup \{q\}$.

Dernièrement, il n'est pas possible de changer le passé, donc la partie de la solution qui précède l'instant de formulation ne peut pas être changée. Soit act une activité de SOL :

CAS 1 act finit avant t. Dans ce cas, $act \in SOL'$.

CAS 2 act commence après t. Dans ce cas, act n'est pas nécessairement dans SOL'.

CAS 3 act commence avant t et finit après t. Dans ce cas, act peut être modifié, en fonction de son type.

— Si act est un trajet à vide, alors $act \in SOL'$.

— Si act est un stationnement, alors il peut être allongé ou raccourci, à condition de toujours finir après t.

— Si act est une charge, alors elle peut être allongée ou raccourcie, à condition de toujours finir après t et d'avoir une durée au moins égale à la durée de manœuvre.

— Si act est une course, alors seules ses portions commençant après t peuvent être modifiées.

L'objectif dans le problème dynamique est d'obtenir la solution à l'instant t_{fin} maximisant la valeur des requêtes traitées.

Conclusion

Ayant modélisé le problème en représentant mathématiquement chacun de ses composants et leurs interactions, il est maintenant possible d'essayer de le résoudre. Deux grandes approches sont présentées dans les chapitres suivants : méthodes exactes et heuristiques.

Chapitre 5

Méthodes exactes et limites

Le problème ayant été modélisé, on peut maintenant aborder le sujet de sa résolution. Dans ce chapitre, deux approches sont étudiées. La première utilise des méthodes d'optimisation largement utilisées, en convertissant le problème en un système d'inéquations linéaires. La seconde consiste à trouver des propriétés qui, si elles existent dans une instance, permettent une résolution exacte en temps polynomial.

5.1 MILP

Les problèmes d'optimisation peuvent souvent être modélisés sous la forme d'un programme linéaire (LP / Linear Program). Tout programme linéaire est composé de :

— Un ensemble de variables $x_1, ..., x_n \in \mathbb{R}^*$.

— un ensemble de contraintes C, chaque contrainte c ayant la forme : $q_1^c x_1 + ... + q_n^c x_n \leq k^c$

— Une fonction objective $f = max\ q_1^f x_1 + ... + q_n^f x_n$

Diverses méthodes existent pour résoudre rapidement et de manière exacte des programmes linéaires, comme la méthode du simplexe [Stone and Tovey, 1991] et celle des points intérieurs [Potra and Wright, 2000]. Par ailleurs, de nombreux logiciels -appelés solveurs- ont été développés, le plus connu et (souvent considéré comme) le plus performant est CPLEX d'IBM [Gearhart et al., 2013].

Les programmes linéaires en nombres entiers (MILP / Mixed Integer Linear Program) sont des variantes dans lesquelles certaines variables sont nécessairement des nombres entiers (généralement, des booléens). Résoudre un MILP est un problème NP-difficile dans le cas général, mais il existe des méthodes qui permettent d'accélérer la recherche d'une solution, et de résoudre de très grandes instances. Diverses méthodes ont été testées pour le TSP et le VRP, comme la méthode des plans sécants (cutting plane) [Dey and Molinaro, 2018] et la méthode de Lin-Kernighan [Papadimitriou, 1992] qui ont permis de résoudre des instances particulières du TSP à plus de 100 000 points de manière exacte. On peut aussi mentionner d'autres méthodes d'optimisation pour le VRP, comme le branch-and-bound, ou la génération de colonnes.

Le modèle présenté dans le chapitre 4 n'est pas un MILP, mais une modélisation en MILP du PCE-ADARP est présentée dans l'annexe A. C'est cette seconde modélisation qui est utilisée dans la présente section.

Le modèle MILP est légèrement différent de celui du chapitre précédent pour une variété de raison : Un MILP est exprimé beaucoup plus simplement en oubliant les concepts d'action et d'activité, et certaine contraintes additionnelles permettent de réduire drastiquement le temps de calcul [1]. Le nombre de charges à chaque chargeur est limité, et il n'est pas possible de se stationner après une course ou une charge, seulement avant. Ces modifications n'ont a priori pas un impact significatif sur les conclusions du présent chapitre [2] et sur la complexité du problème.

5.1.1 Limites de CPLEX - Temps de calcul total

Nous avons mené des expérimentations pour tester les limites d'une résolution passant par un MILP. Les détails de son implémentation sous CPLEX sont également dans l'annexe A.

Dans nos instances de test, le graphe de la ville est générée à partir d'un nombre de requêtes, taxi et chargeur désiré. Pour chaque taxi, chargeur, origine et destination de client, un sommet est créé et placé suivant une loi uniforme dans une zone carrée de 100 mètres sur 100, contenant l'objet associé. La longueur du trajet entre chaque pair de sommets est la distance euclidienne qui les sépare. La vitesse est de 1m/s dans chaque rue, et la congestion est constante. Il y a 3 taxis et 2 chargeurs utilisables 2 fois chacun.

Comme le montre la figure 5.1, le temps de calcul évolue exponentiellement avec le nombre de requêtes. A partir de 15 requêtes, il faut plus d'une heure pour résoudre une instance. La raison principale est qu'avec 15 requêtes, le graphe de la ville contient une quarantaine de sommets, et près de 1600 arcs, chaque arc étant associé (entre autres) à des contraintes de charge, capacité et temps, et des variables de décision correspondantes, auxquelles on ajoute une variable de décision pour chaque arc indiquant s'il est utilisé par un taxi donné. Une instance à 15 requêtes implique donc près de 10000 contraintes et variables de décision. CPLEX est capable de retirer certaines contraintes et variables redondantes à l'aide de précalculs, mais ceux-ci ont leurs limites.

En plus des expérimentations à nombre de requêtes variable de la figure 5.1, nous avons aussi observé le temps de calcul en fonction du nombre de taxis. En doublant le nombre de taxis (6), on observe d'abord que le temps de calcul ne change pas si le nombre de requêtes est inférieur à 9. Arrivé à ce seuil de 9 requêtes, le temps de calcul change drastiquement. Les deux-tiers des instances sont résolus avec une augmentation du temps de calcul comprise entre 10 et 100%. Le dernier tiers en revanche requiert 10 fois plus de temps de calcul. Augmenter le nombre de taxis augmente donc non seulement le temps de calcul [3], mais il augmente aussi massivement la variance du temps de calcul.

5.1.2 Comparaison des temps avec d'autres problèmes

Pour déterminer plus précisément les causes de ces temps de calcul, nous avons progressivement retiré des contraintes du MILP pour le simplifier. Par exemple, en retirant les

1. Le fait qu'un chargeur puisse être utilisé $t_{fin} - t_0$ fois peut poser un problème de taille pour une résolution par un solveur.
2. à condition de supposer qu'une solution optimale a toujours un nombre de charge largement inférieur à $t_{fin} - t_0$
3. sauf dans les cas triviaux , *e.g.* avec une requête par taxi

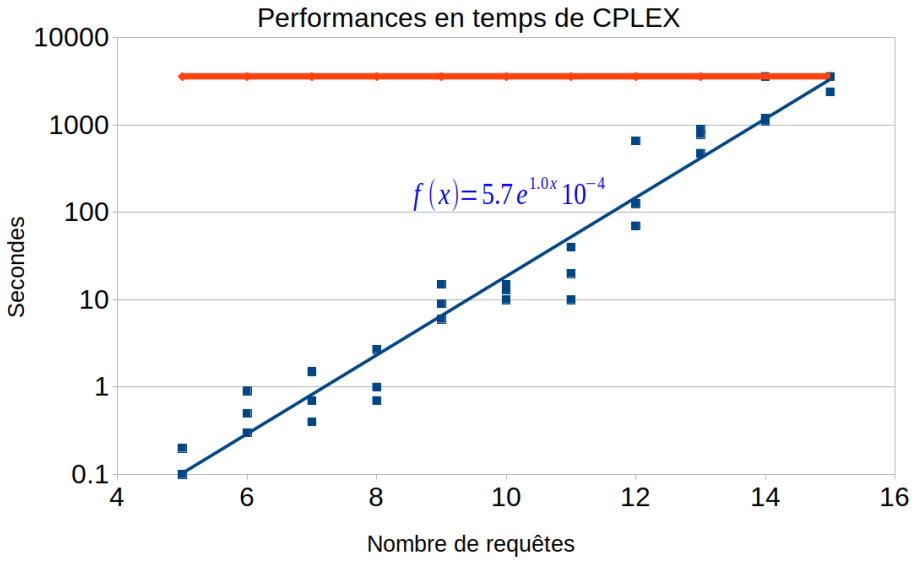


Figure 5.1 – *Chaque point bleu est associé à une unique instance du PCE-ADARP. Les paramètres des instances sont indiqués au début de la section 5.1.1, sauf le nombre de requêtes (abscisse). L'ordonnée indique le temps requis pour trouver la solution optimale. La ligne rouge indique la limite de temps (une heure). La courbe bleue est la régression exponentielle des points bleus.*

Problème	Requêtes	temps
VRPTW	80	0h15
PDPTW	30	0h15
DARP (CPDPTW)	25	0h30
E-ADARP	20	0h30
PCE-ADARP	15	1h00

Table 5.1 – *Temps de calculs nécessaire sur un DELL latitude 7490 pour un problème et un nombre de requêtes donné. Les problèmes sont triés par nombre de contraintes et variables de décisions (Le VRPTW a le moins de contraintes, le PCE-ADARP en a le plus).*

contraintes de batterie, le problème devient similaire mathématiquement à un DARP. Cela nous a permis d'établir pour d'autres problèmes de la littérature le temps de calcul nécessaire lorsqu'on tente de les résoudre sur les mêmes instances avec le même ordinateur (voir table 5.1).

Dans la littérature, des méthodes exactes plus complexes et plus efficaces existent, avec un paramétrage plus fin des variables de CPLEX, ou avec certains pré-calculs adaptés aux instances. Une approche Branch-&-cut pour le E-DARP est proposée par [Bongiovanni et al., 2019], avec des difficultés à traiter plus de 50 clients. Similairement, une méthode exacte de "cut-and-column generation" pour le *Green-VRP* avec durée de recharge fixe est proposée par [Andelmin and Bartolini, 2017], et la table 5.2 présente le temps de calcul annoncé dans cet article pour résoudre des instances en fonction de leur taille.

[Kallehauge, 2006] présente des résultats obtenus avec une approche par génération de colonne sur des variantes des instances de Solomon pour le VRPTW. Bien que plutôt anciens, les résultats obtenus mettent en lumière l'importance des données d'entrées sur la vitesse

Nombre d'arrêts	Nombre de stations	Temps
75	21	1h27
75	28	1h33
98	21	1h27
98	28	1h49
109	21	3h06
109	28	3h08

Table 5.2 – *Chaque ligne indique des caractéristiques d'une instance du G-VRP et le temps qu'il a été nécessaire à [Andelmin and Bartolini, 2017] pour résoudre cette instance.*

Nombre de points de passage	Type	Taux de succès (%)
100	Cluster	100
100	Aléatoire	70
100	Hybride	80
200	Cluster	95
200	Aléatoire	30
200	Hybride	25

Table 5.3 – *Fréquence à laquelle [Kallehauge, 2006] a obtenu la solution optimale en moins d'une heure sur des instances de Solomon, arrondi à 5% près.*

d'exécution de CPLEX.

La table 5.3 présente le taux de succès annoncé dans [Kallehauge, 2006] pour le VRPTW, i.e. la proportion d'instances pour lesquelles la solution optimale a été trouvée en moins d'une heure. Le type d'une instance indique comment les points de passages ont été placés : distribution aléatoire uniforme, par groupe proche géographiquement, ou un mélange des deux.

Lorsque les points sont regroupés, la génération de colonnes est efficace et traite rapidement des instances avec 200 points. mais cette approche devient insatisfaisante dès que les requêtes sont plus dispersées.

Si on observe la différence de temps de calcul et de taille d'instance entre le VRPTW et le PCE-ADARP présentée dans le tableau 5.1, il parait peu probable qu'une variante de cette méthode offre des performances satisfaisantes de manière stable pour résoudre des instances du PCE-ADARP avec plusieurs milliers de requêtes en moins d'une heure, étant donné que les instances réalistes du PCE-ADARP ont probablement une distribution spatio-temporelle hybride des requêtes.

5.1.3 Limites de CPLEX - Temps de calcul pour trouver la solution optimale

Dans la section précédente, les temps de calcul indiquent le temps *total* pris par CPLEX. Ce temps total inclue à la fois le temps pris pour trouver la solution optimale, mais aussi le temps pris pour confirmer que la meilleure solution trouvée est bien optimale. Au début des calculs, CPLEX établit une borne supérieure sur la qualité de la solution. Au fur et à mesure que les calculs avancent et que CPLEX élimine des candidats, cette borne supérieure

diminue. CPLEX confirme que la meilleure solution trouvée est optimale lorsque sa valeur est égale à la borne supérieure.

Dans les faits, on constate que pour le PCE-ADARP, CPLEX trouve la solution optimale relativement tôt, et passe la majeure partie de son temps à confirmer son optimalité. Avec 10 requêtes, la solution optimale est généralement trouvée à la moitié du temps de calcul total. Avec 12 requêtes, au tiers du temps de calcul total. Nous avons testé l'hypothèse que CPLEX trouve une solution proche de l'optimal rapidement avec des dizaines de clients. Des expérimentations avec 100 requêtes et 8 taxis montrent cependant que, sans proposer une solution initiale au problème, il faut plus d'une heure à CPLEX pour trouver une solution de valeur non-nulle pour le PCE-ADARP.

5.1.4 Conclusion sur l'usage d'un MILP

Contrairement aux articles cités ci-dessus, nous avons utilisé CPLEX simplement à partir de notre modèle sous forme de MILP. Nous n'avons pas développé de procédure de *pre-processing* pour tenter de fixer *a priori* la valeur de certaines variables, chercher des coupes spécifiques au problème, ou ajouté une génération de colonne.

Qui plus est, nous aurions pu utiliser un serveur bien plus puissant que la machine sur laquelle nous avons effectué le test. Mais il paraît clair au vu des chiffres présentés dans les 3 tables précédentes que CPLEX ne peut constituer une solution réaliste à la résolution des instances réelles de très grande taille que nous souhaitons résoudre, comptant des milliers, voire des centaines de milliers de requêtes. Pour cette raison, nous avons étudié d'autres pistes.

5.2 Ordre sur les clients

Le PCE-ADARP étant une extension du VRP, et puisque le VRP est NP-difficile dans le cas général, il suit que le PCE-ADARP est au moins NP-difficile, *i.e.* il n'existe pas d'algorithme polynomial connu permettant de le résoudre. Cependant, nous nous concentrons sur des instances proches de celles auxquelles de véritables flottes de taxis autonomes pourraient être confrontées. Ces instances pourraient avoir des caractéristiques communes telles que le PCE-ADARP puisse être résolu en temps polynomial.

Par exemple, les usagers d'un service de taxi ont de petites fenêtres de temps. une personne désirant un taxi à 10h00 sera traité avant une autre désirant un taxi à 11h00, ou alors l'un des deux n'est pas traité. Par conséquent, il peut être relativement rare qu'il y ait une incertitude sur l'ordre de traitement des clients (cette propriété n'est sans doute pas vérifiée pour tous les services de taxis, en particulier dans les villes à très forte densité de clients par kilomètre).

Cette propriété est utilisée par [Bertsimas et al., 2019] pour accélérer la recherche d'une solution par un solveur. Dans le TSP, l'ordre de passage est l'unique difficulté. On peut donc supposer que dans le VRP, qui étend le TSP, si l'ordre de passage est préétabli, alors le VRP est simple à résoudre.

La gestion d'une flotte de taxis peut être modélisée par un VRPTW dynamique. Ayant à notre disposition une flotte de véhicules, on souhaite savoir si un nouveau client peut être traité, en respectant la fenêtre de temps de ce client et de tous les clients déjà acceptés.

On peut voir ce problème comme une succession de VRPTW statique à résoudre. Pour les instances dans lesquelles les fenêtres de temps des clients sont de durée nulle, nous allons montrer qu'il existe un algorithme polynomial capable de les résoudre.

5.2.1 Définition du VRPTW de décision à k véhicules

Dans la suite de cette section, **VRPTW-dk** désigne un VRPTW dans lequel l'objectif est non pas de minimiser la distance totale parcourue, mais plutôt de déterminer s'il est possible de passer par tous les points en respectant les fenêtres de temps avec k véhicules.

Définitions préliminaires

Soit E un ensemble de chemins dans un graphe. On dit que E est un ECC (Ensemble de chemins couvrants) si et seulement si chaque sommet du graphe est présent dans exactement un chemin de E. Un chemin est composé d'au moins 1 sommet et 0 arc. Les chemins d'un ECC sont donc deux-à-deux sommets-disjoints. Un ECCj est un ECC de cardinal j (composé de j chemins).

Données du VRPTW-dk

Soit $G = (V, A, d)$ un graphe complet pondéré avec fenêtres de temps.
Chaque sommet $v \in V$ a une fenêtre de temps $[\alpha_v, \omega_v] \subseteq \mathbb{R}$ durant laquelle le client correspondant doit être traité.
On note $d_a \in \mathbb{R}^+$ le poids de tout arc a, qui est égal à la durée du trajet que l'arc représente.
Soit $k \in \mathbb{N}$ un nombre de véhicules.

Objectif du VRPTW-dk

Déterminer s'il existe un ECCk dans G et un instant de traitement $t_v \in \mathbb{R}$ pour tout sommet $v \in V$ tel que, si l'arc $(u, v) \in A$ est dans un chemin de cet ECCk, alors :

$$max(t_u, \alpha_u) + d_{(u,v)} \leq t_v \leq \omega_v \tag{5.1}$$

5.2.2 Algorithme polynomial exact pour un VRPTW restreint

Définissons T comme étant l'ensemble des instances du VRPTW à fenêtres de temps nulles, donc telles que :

$$\forall\, v \in V,\ \alpha_v = t_v = \omega_v \tag{5.2}$$

Théorème : Le VRPTW-dk est dans P pour toute instance dans T.

On peut supposer que les instances dans T représentent relativement fidèlement des situations auxquels des flottes de taxis réelles sont susceptibles de faire face. Des extensions de ce théorème sont fournies à la fin de ce chapitre.

Preuve : Nous allons présenter un algorithme permettant de résoudre les instances du VRPTW-dk qui sont dans T, puis nous allons montrer que chaque étape de cet algorithme peut être accomplie en temps polynomial.

Voici les principales étapes de cet algorithme. Nous allons construire un DAG (Directed Acyclic Graph / graphe orienté acyclique) G^{DAG} à partir de G, pour ensuite construire un graphe biparti G^{bi} à partir de G^{DAG}. Enfin, nous montrerons qu'il existe un ECCk dans G respectant la contrainte 5.1 si et seulement s'il existe un couplage parfait dans G^{bi}.

Dans une instance de T, les valeurs de α_v, t_v et ω_v sont égales pour tout sommet $v \in V$. On peut donc réécrire la contrainte 5.1 de la manière suivante :

$$\alpha_u + d_{(u,v)} \leq \alpha_v \tag{5.3}$$

On construit $G^{DAG} = (V, A^{DAG})$ un sous-graphe orienté non-pondéré de G. Tout sommet de G est dans G^{DAG}. Les arcs de G^{DAG} sont les arcs de G respectant l'inéquation 5.3, autrement dit :

$$A^{DAG} \subseteq A \tag{5.4}$$

$$\forall\ (u,v) \in A^{DAG} \qquad \alpha_u + d_{(u,v)} \leq \alpha_v \tag{5.5}$$

Montrons que G^{DAG} est bien un DAG. Comme il a été indiqué dans la définition du VRPTW-dk, $\forall\ a \in A$, $d_a \in \mathbb{R}^+$. Avec l'inéquation 5.5, on conclut que $\forall\ (u,v) \in A^{DAG}$, $\alpha_u < \alpha_v$. Par conséquent, il existe un ordre topologique sur G^{DAG}, qui est donc bien un DAG (éventuellement non-connexe). La figure 5.2 présente un exemple de G^{DAG}.

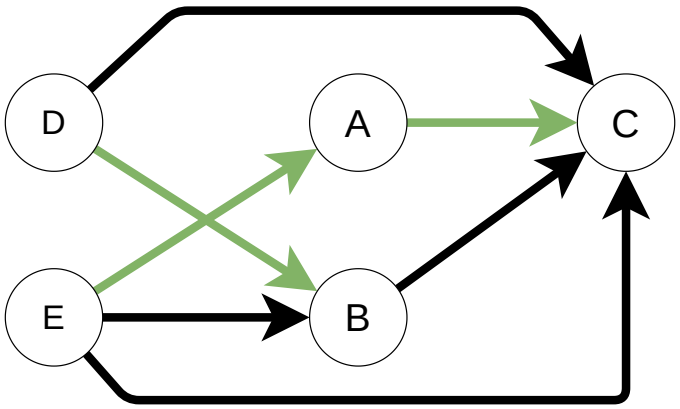


Figure 5.2 – *Un DAG d'un VRPTW-dk à fenêtres de temps nulles, avec 5 clients et deux véhicules. Les arcs verts représentent une solution du problème.*

Notons A^E l'ensemble des arcs qui composent un ECC E. Les arcs de G^{DAG} étant les arcs de G qui respectent l'inéquation 5.3, on conclut que E respecte la contrainte 5.1 si et seulement si $A^E \subseteq A^{DAG}$. Cela nous amène à formuler la propriété suivante.

Propriété 1 : Pour tout ECCk E solution du VRPTW-dk, $A^E \subseteq A^{DAG}$ et E est un ECCk de G^{DAG}.

On construit maintenant un graphe biparti non-orienté $G^{bi} = (V_{in}, V_{out}, A^{bi})$ de la manière suivante. On associe à chaque sommet $v \in V$ un sommet $v_{out} \in V_{out}$ et un sommet $v_{in} \in V_{in}$. De plus, on ajoute à V_{out} un ensemble de sommets X, et on ajoute à V_{in} un ensemble Y. Les ensembles X et Y sont chacun de cardinal k (le nombre de véhicules).

A chaque arc de G^{DAG}, on associe une arête dans G^{bi} (équivalence 5.6). Les sommets de X et de Y sont connectés à tous les sommets associés à un sommet de G^{DAG} (formules 5.7 et 5.8, qui ne changent pas le fait que G^{bi} est biparti). Les arcs de A^{bi} sont donc les suivants :

$$\exists\,(u,v) \in A^{DAG} \quad \Leftrightarrow \quad \exists\,[u_{out}, v_{in}] \in A^{bi} \tag{5.6}$$

$$\forall\, x \in X, v_{in} \in V_{in} \backslash Y \qquad \exists\,[x, v_{in}] \in A^{bi} \tag{5.7}$$

$$\forall\, u_{out} \in V_{out} \backslash X, y \in Y \qquad \exists\,[u_{out}, y] \in A^{bi} \tag{5.8}$$

Montrons qu'il existe un ECCk dans G^{DAG} si et seulement s'il existe un couplage parfait dans G^{bi}. Pour cela, nous allons d'abord montrer que s'il existe un ECCk dans G^{DAG}, alors il existe un couplage parfait dans G^{bi}.

Supposons qu'il existe un ECCk dans G^{DAG} qu'on appelle E.
Définissons $E^{debut} \subseteq V$ et $E^{fin} \subseteq V$ les ensembles de sommets de G^{DAG}, de sorte que $v \in E^{debut}$ si et seulement si v est le premier sommet d'un des chemins de E, et $v \in E^{fin}$ si et seulement si v est le dernier sommet d'un des chemins de E.

On associe arbitrairement à chaque sommet de X un sommet distinct de E^{debut}. Puisque $|E^{debut}| = k = |X|$, cette relation est bijective. On note $v^x \in V$ le sommet de G^{DAG} associé au sommet $x \in X$. De la même manière, on associe arbitrairement à chaque sommet de Y un sommet distinct de E^{fin}, et $|E^{fin}| = k = |Y|$.
Définissons C^{bi} un ensemble d'arêtes de G^{bi} de la manière suivante :

$$\exists\,(u,v) \in A^{E} \Rightarrow \exists\,[u_{out}, v_{in}] \in C^{bi} \tag{5.9}$$

$$x \in X \Rightarrow [x, v^x_{in}] \in C^{bi} \tag{5.10}$$

$$y \in Y \Rightarrow [v^y_{out}, y] \in C^{bi} \tag{5.11}$$

Montrons que C^{bi} est un couplage parfait, *i.e.* un ensemble d'arêtes tel que chaque sommet de G^{bi} est connecté à exactement une arête de C^{bi}.

Commençons par les sommets de V_{out}. Chaque sommet de X est connecté à exactement une arête de C^{bi} dû à l'implication 5.10. Pour tout sommet $u \in V$, soit u est le dernier sommet d'un chemin, et dans ce cas il existe une unique arête de C^{bi} connectée à u_{out} dû à l'implication 5.11, soit u n'est pas le dernier sommet d'un chemin, et il existe une unique arête de C^{bi} connectée à u_{out} dû à l'implication 5.9.

De la même manière, on peut montrer que chaque sommet de V_{in} est connecté à exactement une arête de C^{bi}. Cela nous permet de formuler la propriété suivante.

Propriété 2 : S'il existe un ECCk dans G^{DAG}, alors il existe un couplage parfait dans G^{bi}.

Montrons maintenant que s'il existe un couplage parfait dans G^{bi}, alors il existe un ECCk dans G^{DAG}. Supposons qu'il existe un couplage parfait $C^{bi'}$ dans G^{bi}. On définit d'abord $C^{bi'}_{part}$ un sous-ensemble de $C^{bi'}$ qui ne contient pas les arêtes connectées à un sommet de X ou de Y.
Puisque $|X| = |Y| = k$, et puisqu'il n'existe pas d'arête $[x, y] \in A^{bi}$ avec $x \in X$ et $y \in Y$, on conclut la propriété 3.

Propriété 3 : Il existe k sommets de $V_{in} \backslash Y$ et de $V_{out} \backslash X$ qui ne sont pas connectés à une arête de $C^{bi'}_{part}$.

La manière dont G^{bi} est construit garantit l'égalité 5.12. Puisque $C^{bi'}$ est un couplage parfait, son cardinal est le double de celui de $|V^{bi}|$ on conclut l'égalité 5.13. Avec cette égalité et la propriété 3, on trouve le cardinal de $C^{bi'}_{part}$ (équation 5.14) :

$$|V^{out}| = |V^{in}| = |V| + k \tag{5.12}$$

$$|C^{bi'}| = |V^{out}| \tag{5.13}$$

$$|C^{bi'}_{part}| = |C^{bi'}| - 2 \times k = |V| - k \tag{5.14}$$

Définissons $A^{E'}$ comme l'ensemble des arcs de G^{DAG} associés à des arêtes de $C^{bi'}_{part}$ dans le DAG :

$$\exists\, (u, v) \in A^{E'} \Leftrightarrow \exists\, [u_{out}, v_{in}] \in C^{bi'}_{part} \tag{5.15}$$

$$|A^{E'}| = |C^{bi'}_{part}| = |V| - k \tag{5.16}$$

Montrons que $A^{E'}$ forme un ensemble de chemins sommet-disjoints. Chaque sommet de V_{out} et V_{in} est connecté à au plus une arête de $C^{bi'}_{part}$, donc avec l'équivalence 5.15, on déduit que chaque sommet $v \in V$ a au plus un arc entrant et un arc sortant dans $A^{E'}$. Puisque de plus G^{DAG} est un DAG, on déduit que $A^{E'}$ ne contient pas de circuit, et donc que $A^{E'}$ est un ensemble de chemins deux-à-deux sommets-disjoints.

Définissons E'_{part} comme étant l'ensemble de chemins deux-à-deux sommets-disjoints formés par les arcs de $A^{E'}$.

Puisque $A^{E'}$ ne contient pas d'arc dont un sommet est dans X ou Y, si $[x, v_{in}]$ et $[v_{out}, y]$ sont dans $C^{bi'}$ avec $x \in X, y \in Y$, et $v \in V$, alors $v \notin E'_{part}$. Définissons $E'_{part,2}$ comme l'ensemble des chemins tel que, avec $v \in V$ un sommet, si $v \notin E'_{part}$, alors il existe dans $E'_{part,2}$ un chemin dont le seul sommet est v.

Définissons l'ensemble de chemins $E' = \{E'_{part}, E'_{part,2}\}$. Par construction, E' est un ECC. Montrons qu'il s'agit d'un ECCk.

Puisque les chemins de $E'_{part,2}$ ne contiennent aucun arc, et puisque les chemins de E'_{part} contiennent $|A^{E'}| = |V| - k$ arcs (*cf.* équation 5.16), alors E' contient $|V| - k$ arcs. Avec $l > 0$, un chemin contenant $l - 1$ arcs couvre l sommets. Par extension, un ensemble de k chemins sommet-disjoints contenant $|V| - k$ arcs couvre $|V|$ sommets, donc E' est un

ECCk. Cela nous permet de formuler la propriété 4.

Propriété 4 : S'il existe un couplage parfait dans le graphe biparti, il existe un ECCk dans le DAG.

À partir des propriétés 2 et 4, on conclut la propriété 5.

Propriété 5 : Il existe un couplage parfait dans le graphe biparti si et seulement si il existe un ECCk dans le DAG.

À partir des propriétés 1 et 5, on conclut que C existe si et seulement s'il existe un couplage parfait dans le graphe biparti, et donc que la solution du problème pour une instance donnée est "vrai" si et seulement s'il existe un couplage parfait dans le graphe biparti correspondant à cette instance.

Complexité de l'algorithme

Voici un résumé de l'algorithme de résolution du VRPTW-dk sur un graphe dans T.

— On génère un graphe G^{DAG} qui est une copie de G de laquelle certains arcs ont été supprimés, le transformant en DAG.

— On génère un graphe biparti G^{bi} à partir de G^{DAG}.

— On trouve un couplage parfait dans G^{bi}, avec par exemple l'algorithme hongrois (de Kuhn-Munkres) [Kuhn, 1955].

— A partir de ce couplage parfait, on peut trouver un ECCk dans G^{DAG} et dans G.

En termes de complexité, déterminer si une instance est dans T est polynomial, chaque génération de graphe est polynomial, trouver un couplage parfait dans un graphe biparti est polynomial, et la génération d'un chemin dans G à partir d'un couplage dans G^{bi} est polynomial.

Il existe donc bien un algorithme polynomial permettant de résoudre le VRPTW de décision à k véhicules, dans une instance dans laquelle les fenêtre de temps sont de taille nulle et aucun arc n'est de durée nulle.

C.Q.F.D.

5.2.3 Application au PCE-ADARP

Montrons que l'algorithme présenté ici ne peut pas être adapté pour traiter des instances réalistes du PCE-ADARP, à cause des contraintes liées à la batterie, au retard, et au partage de course.

Batterie

Considérons un VRPTW-dk sur un graphe dans T, dans lequel les taxis ont chacun une batterie ayant une même taille m qu'ils ne peuvent pas recharger. Ce problème revient à trouver un ECCk dans G^{DAG} tel qu'aucun chemin n'a un poids total supérieur à m. Ce problème est appelé *MAXLENGTH-COVER* par [Ntafos and Gonzalez, 1984], qui prouve qu'il est NP-complet.

Retard

Considérons un VRPTW-dk sur un graphe tel que les clients tolèrent un retard ($t_v - \alpha_v$ peut être positif), mais ce retard est suffisamment petit pour qu'il y ait un ordre topologique sur les sommets. Autrement dit, avec A^{DAG} l'ensemble des arcs susceptibles d'être dans la solution, on a :

$$\forall (u,v) \in A^{DAG} \qquad \alpha_u + d_{(u,v)} \leq \omega_v \;\; \Rightarrow \;\; \alpha_v + d_{(v,u)} > \omega_u \tag{5.17}$$

Notons $r(u,v)$ le retard tel que $\alpha_u + d_{(u,v)} = \alpha_v + r(u,v)$, et prenons pour exemple une instance dans laquelle on ne peut jamais arriver en avance ($\forall\ (u,v) \in A$, on a $r(u,v) \geq 0$). Dans ce cas, si un chemin d'une solution est (u,v,w), alors on a $t_w = \alpha_w + r(u,v) + r(v,w)$.

Pour cette raison, dans une instance dans laquelle le retard maximal toléré est m et dans laquelle il n'est jamais possible d'arriver en avance, trouver une solution revient fondamentalement à trouver un ECCk dans G^{DAG} tel qu'aucun chemin n'a un poids total supérieur à m. On retrouve donc le MAXLENGTH-COVER.

Partage de course

Le partage de course implique nécessairement une tolérance au retard, ce qui signifie un problème NP-complet, comme on vient de le voir.

Le partage de course implique également des contraintes de capacité, ce qui fait qu'on a encore une fois une réduction polynomiale de MAXLENGTH-COVER.

En mettant de côté ces deux difficultés, et en considérant simplement un PDP à capacité infinie dans un DAG. Autrement dit, avec k taxis, on cherche un ensemble sommet-couvrant de k chemins dans un DAG, tel que certaines paires de sommets (origine/destination) font partie du même chemin. Ce problème est appelé Must PaiR problem (MPR) par [Ntafos and Gonzalez, 1984], qui prouve qu'il est NP-complet.

5.2.4 Conclusion

Même en simplifiant le PCE-ADARP en supposant un ordre de traitement des clients prédéterminé, il reste impossible à résoudre avec un algorithme polynomial, pour au moins 4 raisons différentes : batterie limitée (même sans chargeurs), retard, partage de course (même avec capacité infinie), et capacité finie des taxis. Pour résoudre ce problème de manière exacte avec une approche algorithmique, il faudrait donc trouver des caractéristiques des instances dans le monde réel qui permettent de rendre facile à la fois la gestion de la batterie, le retard, le partage de course, et la capacité des taxis, ou bien accepter une perte significative de réalisme des instances et du modèle.

Considérant le nombre de points problématiques, cette approche semble peu intéressante. Et comme il a été observé plus tôt dans ce chapitre, des méthodes exactes plus classiques, passant par une modélisation en MILP font également face à des difficultés importantes.

Au final, il paraît donc judicieux d'envisager l'usage d'heuristiques qui considèrent le problème dans sa globalité. C'est cette considération qui nous amène à l'algorithme glouton et les méthodes de recherche de voisinage présentés dans le chapitre suivant.

Chapitre 6

Algorithme glouton

6.1 Considérations générales

Dans le chapitre précédent, nous avons constaté que des méthodes de programmation linéaire ou des algorithmes exacts ont peu de chance de produire des résultats satisfaisants sur des instances ayant les dimensions qui nous intéressent. Pour cette raison, nous avons choisi de nous concentrer sur des approches heuristiques et méta-heuristiques.

Dans le problème dynamique, une solution initiale est générée, puis elle est complétée par un algorithme glouton à chaque formulation de requête.

Dans le problème statique, deux étapes sont ajoutées. Après la génération d'une solution par l'algorithme glouton, cette solution est améliorée par un algorithme de montée de colline, suivi par un recuit simulé.

Dans la solution initiale, aucune requête n'est traitée. Chaque taxi est stationné du début de son edtt (t_0) jusqu'à la fin (t_{fin}). Une telle solution n'est pas nécessairement admissible, car le niveau de batterie initial de certains taxis peut être inférieur au niveau de batterie final minimal autorisé (w_{fin}).

La méthode employée pour l'insertion de charges dans la solution initiale est la même que la méthode employée pour insérer des charges durant l'exécution de l'algorithme glouton. Plutôt que de présenter maintenant cette méthode, présentons d'abord la manière dont l'algorithme glouton fonctionne, en ignorant les contraintes liées aux batteries des taxis.

6.1.1 Fonctionnement général de l'algorithme

L'algorithme glouton est composé de deux grandes parties : l'insertion de requêtes et l'insertion de charges. Son principe général est le suivant. Insérer les requêtes une par une, et après chaque insertion de requête, insérer des charges.

Durant l'exécution de cet algorithme, une solution inadmissible n'est jamais conservée. Autrement dit, si l'insertion d'une requête ou des charges rend la solution inadmissible, l'insertion de cette requête et de ces charges est annulée, et on passe à l'insertion de la requête suivante.

Cet algorithme est glouton, dans la mesure où une insertion de requête dans un edtt est définitive. En revanche, les charges peuvent être supprimées et déplacées librement. La complexité algorithmique de chaque partie de cet algorithme est étudié dans les dernières

sections de ce chapitre, et des améliorations sont également présentées pour réduire le temps de calcul.

6.1.2 Définitions

Avant de présenter l'algorithme glouton, voici quelques rappels et définitions qui seront utiles dans ce chapitre et le suivant.

Notons $arrive_a(t)$ l'instant de fin de la traversée d'un arc a commençant à l'instant t. Notons $depart_a(t)$ l'instant de début de la traversée d'un arc a pour la finir le plus tard possible avant t (6.1). On peut constater que, si la valeur de congestion de la ville est constante, alors la fonction $arrive_a$ est bijective, et $depart_a$ est son inverse (6.2).

$$depart_a(t) = max\{t' \in T \mid \ arrive_a(t') \leq t\} \qquad \forall\ a \in A, t \in T \tag{6.1}$$

$$t = depart_a(arrive_a(t)) \qquad \text{Si } congestion_t = congestion_{t'} \forall\ t, t' \in T \tag{6.2}$$

Compatibilité : Deux activités sont compatibles si elles peuvent être dans le même edtt. Soit a et c deux activités, on note $arrive(a, c)$ l'instant d'arrivée à l'origine de c en partant de la destination de a à l'instant où a finit, autrement dit :

$$arrive(a, c) = arrive_{(destination(a), origine(c))}(fin(a))$$

Dans les instances à congestion constante, on dit que a et c sont compatibles si et seulement si on peut commencer l'une après avoir fini l'autre :

$$arrive(a, c) \leq debut(c) \text{ ou} \tag{6.3}$$

$$arrive(c, a) \leq debut(a) \tag{6.4}$$

Dans les instances à congestion variable, cette définition est étendue. Soit b une activité d'un edtt e telle que :

$$arrive(b, c) \leq debut(c) \tag{6.5}$$

b est donc compatible avec c, et b précède c. Dans ce cas, toute activité $a \in e$ précédant b est aussi considérée comme compatible, même si $arrive(a, c) > debut(c)$ et $arrive(c, a) > debut(a)$. De même, si $arrive(c, b) \leq debut(b)$, alors toute activité $a \in e$ suivant b est aussi considérée comme compatible.

Compatibilité partielle : Une charge ou une phase de stationnement a est partiellement compatible avec une activité b si a peut être raccourcie suffisamment pour devenir compatible avec b. Enlever les activités d'un edtt incompatibles avec une activité b signifie ne garder que les activités compatibles, et les activités partiellement compatibles raccourcies.

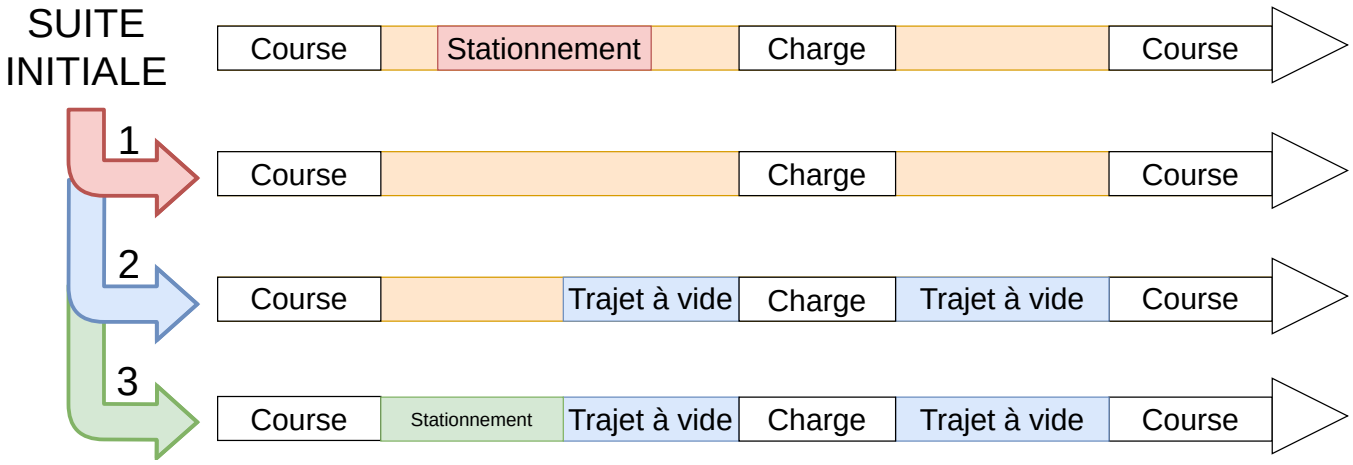


Figure 6.1 – *Contiguïsation d'une suite d'activité. La suite d'activité initiale (en haut) contient des trous (en jaune). 1) Les phases de stationnement et trajets à vide connectés à ces trous (en rouge) sont retirés. 2) Dans les trous, des trajets à vide (en bleu) commençant le plus tard possible sont insérés pour garantir la contiguïté spatiale. 3) Dans les trous, des phases de stationnement (en vert) sont insérées pour garantir la contiguïté temporelle.*

Intervalle modifiable : Un intervalle modifiable $\langle I \rangle$ d'un edtt est une portion de cet edtt (donc une suite contiguë d'activités) dans laquelle les activités peuvent être modifiées. Par exemple, si on autorise de modifier les charges, les phases de stationnement et les trajets à vide, mais pas les courses, alors un intervalle modifiable est une portion d'edtt qui ne contient pas de course. Un intervalle modifiable est toujours maximal, donc par exemple, si les trajets à vide peuvent être modifiés, alors l'activité précédant ou suivant l'intervalle modifiable dans l'edtt ne peut pas être un trajet à vide. Toutefois, dans le problème dynamique, cet intervalle modifiable ne peut pas concerner le passé.

Début d'intervalle : Dans un intervalle modifiable $\langle I \rangle$ d'un edtt e, un début d'intervalle désigne l'activité de e qui précède i_1. On définit similairement une fin d'intervalle. Pour tout intervalle modifiable $\langle I \rangle$, on note i_0 et i_{-0} (ou $i_{|\langle I \rangle|+1}$) les activités précédant et suivant cet intervalle sans en faire partie (rappel sur la notation des ensembles orientés au début du chapitre 4). Si i_1 est la première activité de l'edtt, alors i_0 est une activité artificielle telle que $fin(i_0) = t_0$ et $destination(i_0) = origine(i_1)$. De même, si i_{-1} est la dernière activité de l'edtt, alors $fin(i_{-0}) = t_{fin}$, et $origine(i_{-0}) = destination(i_{-1})$.

Contiguïser : Soit $\langle I \rangle$ une suite d'activités (par exemple, un intervalle modifiable ou un edtt). On dit que $\langle I \rangle$ est incomplète si elle contient des intervalles de temps durant lesquels le taxi n'est associé à aucune activité, *i.e.* tel que $\exists\, k \in \mathbb{N}^+$ pour lequel $fin(i_k) \neq debut(i_{k+1})$ ou $destination(i_k) \neq origine(i_{k+1})$. Contiguiser cette suite d'activités signifie retirer les phases de stationnement et les trajets à vide, puis en insérer de nouveaux, de sorte que la suite d'activités résultante soit contiguë spatialement et temporellement (définition de la contiguïté à la section 3.3.1). La figure 6.1 présente un exemple de cette opération, qui peut échouer et renvoyer une erreur, si un trajet à vide inséré est plus long que l'intervalle dans lequel on l'insère.

6.2 Présentation de l'algorithme glouton, sans charges

Dans cette section, l'algorithme glouton est présenté en commençant par ses petits composants pour finir par sa structure globale. Présentons d'abord la génération d'une course à partir d'une requête, puis l'insertion d'une course dans un edtt. La méthode d'insertion de charges est présenté dans une des sections suivantes.

6.2.1 Création d'une course traitant la requête

Pour insérer une requête dans un edtt, la première étape consiste à générer une course traitant cette requête. Trois approches sont possibles.

Création d'une course individuelle

La première méthode consiste simplement à créer une course ne traitant que la requête et commençant à l'instant idéal de départ du client.

Création simple d'une course partagée

La deuxième approche consiste à créer une course partagée en insérant la requête dans une course déjà existante.

En résumé, la requête est insérée dans la course de sorte que la course soit modifiée le moins possible, que sa durée totale après insertion soit minimisée, que les contraintes de temps de tous les clients soient respectées, de même que les contraintes de capacité du taxi. Notons que le problème qui consiste à trouver un ordre dans lequel les arrêts d'une course doivent avoir lieu pour minimiser la distance parcourue est un DARP, et est donc NP-difficile.

Soit q une requête et c une course. Pour rappel, $\langle P^c \rangle$ désigne la suite des portions de c (*cf.* section 4.2.3).

Insérer q dans c signifie créer une course c' telle que $Q(c') = Q(c) + q$. Supposons que la montée de q a lieu dans la portion i de c' et que la descente de q a lieu dans la portion j de c'. Autrement dit, $v(p_i^{c'}) = s_q^+$ et $v(p_j^{c'}) = s_q^-$. Pour que l'insertion soit valide, l'ensemble des conditions suivantes doivent être respectées.

L'ordre relatif des montées et descentes déjà existantes n'est pas modifié.

$$v(p_k^c) = v(p_k^{c'}) \qquad \forall k \mid\ k < i \tag{6.6}$$

$$v(p_k^c) = v(p_{k+1}^{c'}) \qquad \forall k \mid\ i < k < j \tag{6.7}$$

$$v(p_k^c) = v(p_{k+2}^{c'}) \qquad \forall k \mid\ j < k \tag{6.8}$$

Si la montée de q n'est pas la première gestion de stop de c', alors l'instant de début de la course est inchangé. Sinon, cet instant est avancé, mais le moins possible.

$$i > 1 \Rightarrow debut(c') = debut(c) \tag{6.9}$$

$$i = 1 \Rightarrow debut(c') = max(\alpha(s_q^+), debut(c) - depart_{(p_1^{c'}, p_2^{c'})}(debut(c))) \tag{6.10}$$

La nouvelle course c' doit également respecter toutes les contraintes que doit respecter une course quelconque. Le taxi ne peut notamment jamais être vide durant la course, donc la position $i = 1$ et $j = 2$ est invalide, de même que la position $i = |P^c| + 1$ et $j = |P^c| + 2$.

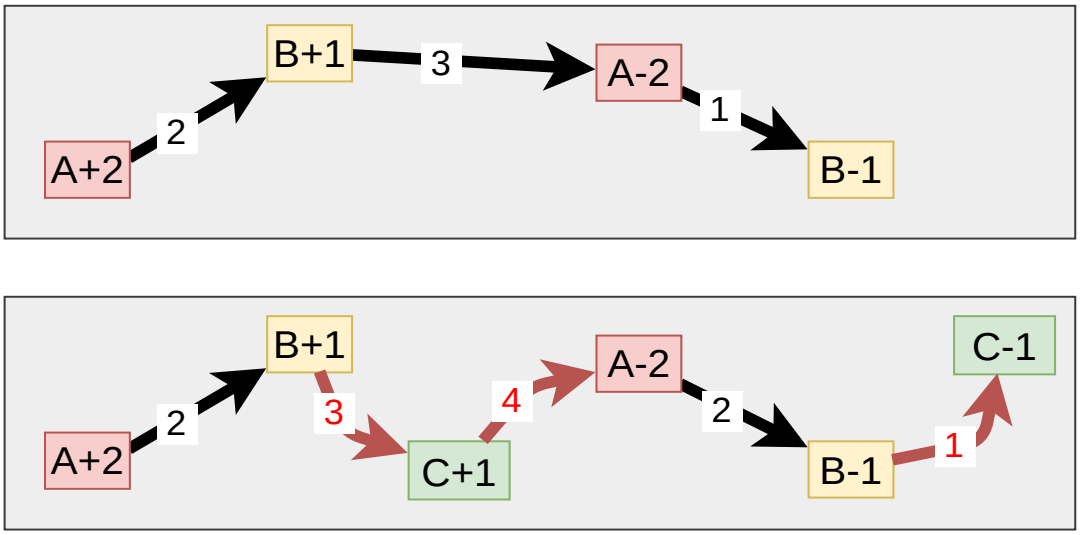


Figure 6.2 – *Une course, avant et après l'insertion d'une requête C. "A+2" représente la montée du client A (2 passagers) et "A-2" représente sa descente. Le chiffre sur un arc indique le nombre total de passagers dans le taxi.*

Il peut exister plusieurs valeurs de c' respectant les contraintes susmentionnées pour une paire de positions (i, j) donnée. Parmi ces valeurs, on choisit toujours celle qui minimise la durée de la course ($duree(c')$). L'algorithme permettant d'insérer une requête dans une course consiste simplement à générer c' pour toutes les positions de i et j avec $i < j$, et choisir parmi les candidats la course de durée minimale.

La figure 6.2 présente l'insertion d'une requête dans une course respectant toutes ces contraintes, et minimisant la durée totale de la course.

Création d'une course par fusion

La troisième méthode pour insérer une requête dans un edtt consiste à l'insérer dans une paire de courses qu'on fusionne. La montée du client est insérée dans la première course, et sa descente est insérée dans la seconde. La figure 6.3 montre un exemple d'une telle insertion.

Plus formellement, soit c_1 et c_2 deux courses, telles que $fin(c_1) \leq debut(c_2)$ et q la requête qu'on veut insérer. L'insertion avec fusion consiste à générer une course c' telle que $\langle P^{c'} \rangle$ est la concaténation de $\langle P^{c_1} \rangle$ et $\langle P^{c_2} \rangle$, puis à insérer la requête q avec la méthode présentée dans la sous-section précédente (et encore une fois, avec la contrainte que le taxi ne doit jamais devenir vide entre $p_1^{c'}$ et $p_{-1}^{c'}$.

Conclusion sur la génération de course

Toutes les méthodes pour générer une nouvelle course à partir d'une requête ont été présentées. Dans le cadre de l'algorithme glouton, ces trois méthodes sont utilisées pour chaque requête. Avant de préciser dans quel ordre et sous quelles conditions, montrons d'abord ce qui peut être fait avec une course ainsi générée.

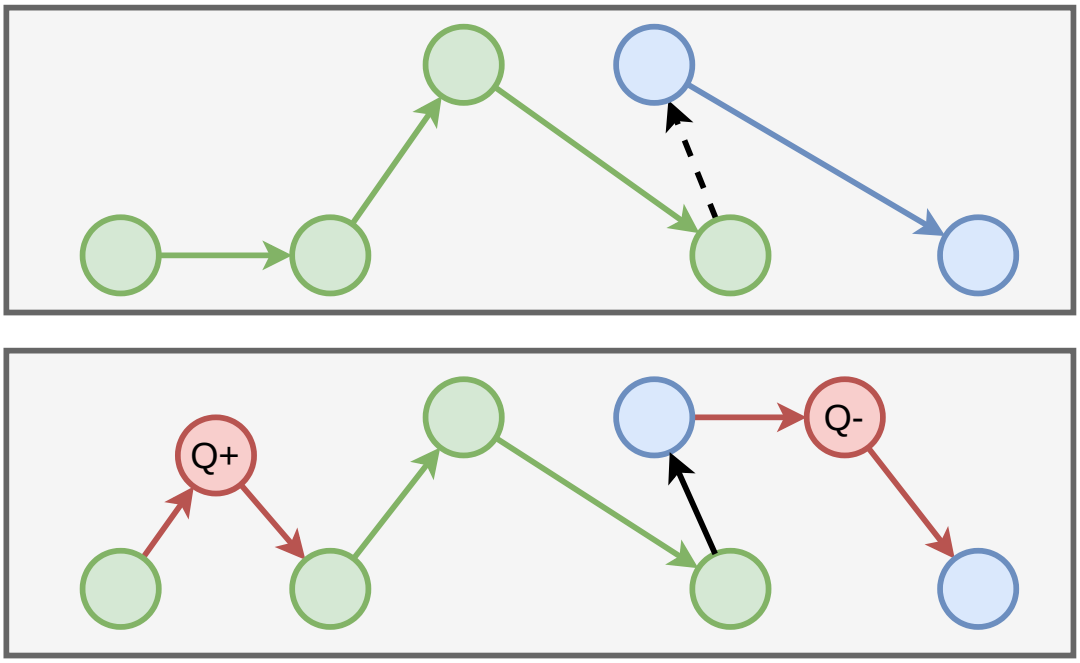


Figure 6.3 – *En haut une course partagée (vert) suivie d'une course individuelle (bleue). En bas, le résultat de l'insertion de la requête q avec fusion des courses ($Q+$ est la montée du client et $Q-$ sa descente).*

6.2.2 Insertion d'une course dans un intervalle modifiable

Montrons maintenant comment une course quelconque peut être insérée dans un edtt. Pour cela, il faut trouver un intervalle modifiable dans lequel la course peut être insérée. Une insertion de course dans un intervalle $\langle I \rangle$ se fait en trois temps. Premièrement, établir un instant de début pour la course, de sorte que la course est compatible à la fois avec i_0 et i_{-0}. Deuxièmement, retirer les activités de $\langle I \rangle$ incompatibles avec la course, puis insérer la course dans le vide généré. Troisièmement, contiguïser $\langle I \rangle$ pour combler le vide restant. L'algorithme 1 présente plus en détail ce processus, et la figure 6.4 présente un exemple d'insertion dans un intervalle modifiable.

6.2.3 Insertion de course dans un edtt

Insérer une course c dans un edtt e se fait en deux étapes. Premièrement les intervalles modifiables sont établis. Dans le cadre de l'algorithme glouton, il existe un intervalle modifiable entre chaque paire de courses successives de l'edtt, en ignorant toute course contenant une requête traitée par c. Autrement dit, chaque activité act d'un intervalle respecte l'une des deux contraintes suivantes :

$$type(act) \neq \text{course} \tag{6.11}$$
$$type(act) = \text{course, et } Q(act) \cap Q(c') \neq \emptyset \tag{6.12}$$

Une fois tous les intervalles modifiables établis, la course est insérée dans le premier intervalle modifiable pour lequel l'insertion est possible, en utilisant l'algorithme 1. Cette opération d'insertion dans un edtt renvoie une erreur si aucun intervalle modifiable ne le permet.

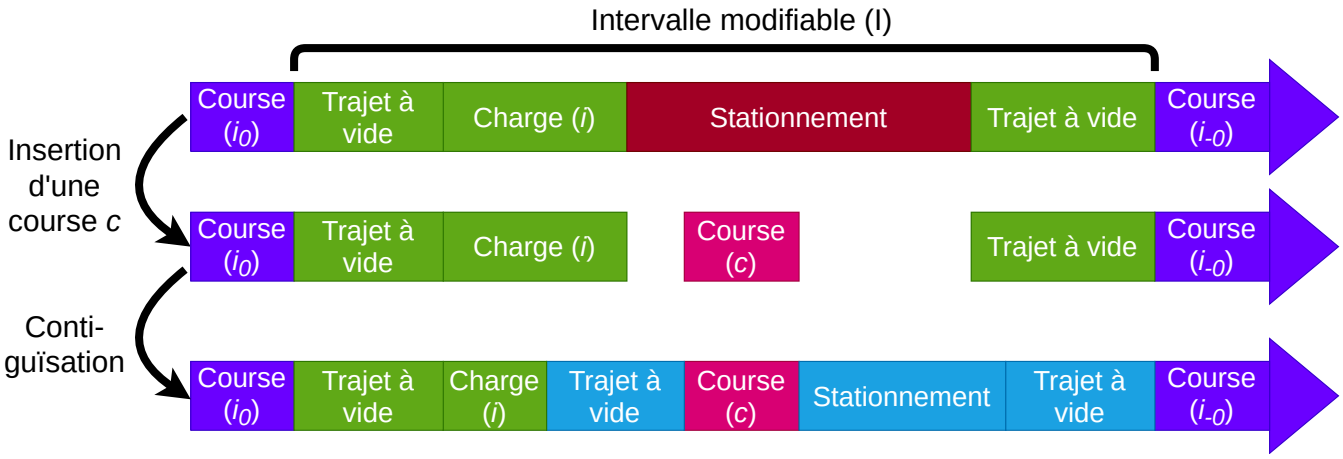


Figure 6.4 – *Un intervalle modifiable I, l'activité i_0 qui le précède et l'activité i_{-0} qui le suit. Une course c (en rose) est insérée dans I. Les activités incompatibles avec c (en rouge) sont retirées. I est ensuite contiguïsé, éventuellement en raccourcissant une activité partiellement compatible (ici, la charge qui précède la course).*

algorithme 1 – Insertion d'une course dans un intervalle modifiable

ENTREE : Une course c, un intervalle modifiable $\langle I \rangle$
SORTIE : Un intervalle modifiable, ou une erreur

DEBUT :
Soit i la dernière activité de $\langle I \rangle \cup i_0$ au moins partiellement compatible avec c et finissant avant c. Dans la figure 6.4, il s'agit de la charge.
si un tel i n'existe pas (par exemple, c commence avant i_0) **alors**
  Retarder l'instant de début de c dans le temps, jusqu'à ce que c devienne compatible avec i_0.
  si les contraintes de temps de c ne sont plus respectées **alors**
    renvoie ERREUR
Retirer de $\langle I \rangle$ toutes les activités incompatibles avec c
Insérer c dans $\langle I \rangle$
Contiguïser $\langle I \rangle$. Peut renvoyer une erreur (voir section 6.1.2).
renvoie $\langle I \rangle$

6.2.4 Insertion de requête dans un edtt

Nous avons établi les différentes méthodes pour créer une course à partir d'une requête, puis l'algorithme qui insère une course dans un edtt. On peut maintenant présenter la manière dont une requête q est insérée dans un edtt e.

Soit $\langle C^e \rangle$ la liste des courses de e. La première étape de l'algorithme d'insertion de requête dans un edtt consiste à lister toutes les courses pouvant être générées à partir de q et de $\langle C^e \rangle$, en utilisant les trois méthodes présentées dans la section 6.2.1. Cette liste l contient donc :

— une course individuelle traitant q

— pour chaque course $c \in \langle C^e \rangle$, une course partagée obtenue en insérant q dans c avec la méthode de création simple

— pour tout $i \in \mathbb{N}^+$, la course obtenue par fusion avec les courses c_i^e et c_{i+1}^e

On établit une seconde liste l' contenant l'ensemble des edtt pouvant être obtenus en insérant une des courses de l dans e avec la méthode de la section 6.2.3.

Si l' est vide, c'est qu'il n'existe pas de course de l qui peut être insérée dans e sans générer d'erreur. Dans ce cas, l'insertion de q dans e échoue. Sinon, parmi les edtt de l', on choisit celui qui minimise la distance totale parcourue par le taxi, et on remplace e par cet edtt dans la solution.

6.2.5 Algorithme complet

On a maintenant tous les éléments requis pour décrire comment une requête est insérée dans l'algorithme glouton, en ignorant les contraintes liées aux batteries des taxis. Notons que, même sans ces contraintes, le problème reste au moins NP-complet dans la mesure où il étend le DARP.

La première étape de l'algorithme consiste à trier les requêtes pour décider de leur ordre d'insertion. Dans le problème dynamique, les requêtes sont obligatoirement triées par instant de formulation. Dans le problème statique, n'importe quel ordre peut être choisi.

A chaque itération, on s'occupe de la première requête qui n'a pas encore été vue. Pour insérer une requête q dans une solution, on crée une liste qui contient, pour chaque edtt, la variante de cet edtt généré avec la méthode d'insertion de requête dans un edtt de la section 6.2.4, si l'insertion est possible.

Dans cette liste, on choisit un edtt (taxi) à l'aide d'un critère qui est un des paramètres de l'algorithme glouton. On peut par exemple choisir d'envoyer le taxi le plus proche, *i.e.* dont l'edtt minimise la durée du déplacement qui précède la montée du client inséré.

L'ordre d'insertion des requêtes et le choix du taxi sont les seuls paramètres de l'algorithme glouton, en excluant les paramètres associés à l'insertion de charges.

6.2.6 Garantie sur la qualité de la solution

Sans contraintes de charge, une solution viable est toujours trouvée, et si une requête peut être traitée, alors la solution trouvée par le glouton contient au moins une requête traitée.

Par exemple, avec 1 taxi, le glouton insère nécessairement dans son edtt la première requête qu'il est capable de traiter. La valeur de cette solution est au moins $\frac{1}{|Q|-1}$ la valeur de la solution optimale si on trie les requêtes par valeur de manière décroissante (les requêtes de plus forte valeur sont insérées en premier).

Dans cet exemple, si toutes les autres requêtes sont incompatibles avec l'unique insérée dans l'edtt de ce taxi, si toutes ces requêtes sont compatibles entre elles (il peut exister un edtt les contenant toutes), et si toutes les requêtes sont de même valeur, alors la valeur de la solution trouvée par l'algorithme glouton est exactement $\frac{1}{|Q|-1}$ fois la valeur de la solution optimale.

Avec $|Q|$ infini, on a donc une approximation non-bornée, même sans contraintes de charge.

6.3 Sous-problème de génération de charges (PGC)

Toutes les étapes de l'algorithme glouton ont été présentées, à l'exception de la partie consacrée aux contraintes de batterie.

La génération de charge à lieu durant l'exécution de la méthode de la section 6.2.4 (Insertion de requête dans un edtt). Après avoir généré un nouvel edtt en insérant une requête dans un edtt existant, il est systématiquement vérifié que les contraintes de batteries sont respectées, et si elles ne sont pas respectées, des charges sont insérées dans l'edtt.

L'insertion de requête et l'insertion de charges sont donc traitées séparément. Avant de développer une méthode pour insérer les charges, nous avons déterminé la complexité du sous-problème suivant : "Ayant des edtt qui contiennent des courses qu'on ne peut pas retirer ou déplacer, existe-t'il une attribution des chargeurs respectant toutes les contraintes de charge". Appelons ce problème PGC (Problème de Génération de Charges).

Dans cette section, nous allons montrer que PGC est au moins fortement NP-complet, même dans le cas particulier où les taxis ne traitent aucun client.

6.3.1 Modélisation de PGC

Simplifications et définitions

Les taxis ont tous la même consommation énergétique par mètre et par seconde, la valeur de congestion de la ville est constante, la recherche de place de stationnement est finie instantanément et ne consomme pas d'énergie, l'opération de connexion à un chargeur ne consomme pas d'énergie. Si un chargeur est disponible à un instant t, alors tous les chargeurs sont disponibles à cet instant.

On définit un intervalle modifiable i par un point de départ dans le temps ($debut(i) \in \mathbb{N}$) et l'espace ($origine(i) \in V$), un point d'arrivée dans le temps ($fin(i)$) et l'espace ($destination(i)$), et un intervalle de charge $[w^i_{min}, w^i_{max}] \subseteq \mathbb{R}_{[0,+\infty]}$ qui indique la quantité d'énergie qui peut/doit être chargée avant fin_i.

Données du problème

On a un graphe orienté complet $G = (V, A)$, dans lequel chaque arc $(u, v) \in A$ a une consommation énergétique $w(u, v) \in \mathbb{R}$ et une durée $duree(u, v) \in \mathbb{N}$.

On a un ensemble de taxis X. Chaque taxi x est associé à une suite d'intervalles modifiables $\langle I^x \rangle$.

On a un ensemble de chargeurs $\mathfrak{C}$. Chaque chargeur $\mathfrak{c}$ est associé à une puissance $w_{\mathfrak{c}}$ et un lieu $v_{\mathfrak{c}} \in V$. Les chargeurs ont une suite d'intervalles de temps durant lesquels ils sont tous disponibles $\langle Dispo \rangle$. La durée de l'opération de connexion/déconnexion d'un taxi à un chargeur est $manoeuvre \in \mathbb{N}$.

Solution du problème

Une charge c est associée à un taxi $taxi(c)$, un chargeur $\mathfrak{c}(c) \in \mathfrak{C}$, un instant initial $debut(c) \in \mathbb{N}$ et un instant final $fin(c) \in \mathbb{N}$. On note $v(c) = v_{\mathfrak{c}}$.

Dans la suite de cette section, la notation $\langle C^z \rangle$ désigne l'ensemble des charges associé à l'objet z, trié par instant de début. Par exemple, si z est un taxi, alors $\langle C^z \rangle$ est la suite des charges $\langle C^z \rangle = \langle c \in C \mid \ taxi(c) = z \rangle$. De même, $C^{y,z} = C^y \cap C^z$. La fonction $duree\_utile(c) = fin(c) - debut(c) - manoeuvre$ indique la durée utile d'une charge c, pendant laquelle un taxi gagne de l'énergie.

Une solution est un ensemble de charges C, tel que, pour tout intervalle modifiable d'un taxi, la quantité d'énergie chargée avant la fin de cet intervalle par le taxi est comprise dans l'intervalle de charge de l'intervalle modifiable. L'objectif est donc de déterminer s'il existe une solution telle que :

$$w^i_{min} \leq \sum_{c \in C^x \mid \ fin(c) \leq fin(i)} duree\_utile(c) \times w_{\mathfrak{c}(c)} \leq w^i_{max} \qquad \forall\, x \in X, i \in \langle I^x \rangle \tag{6.13}$$

Une solution doit respecter les contraintes suivantes.

Contraintes

Il y a deux types de contraintes : de temps, et d'énergie.

Une charge ne peut avoir lieu que durant un intervalle de temps durant lequel les chargeurs sont utilisables.

$$debut(dispo) \leq debut(c) < fin(c) \leq fin(dispo) \qquad \forall\, c \in C, \exists!\ dispo \in \langle Dispo \rangle \tag{6.14}$$

Un chargeur ne peut être utilisé que par un taxi à la fois.

$$fin(c_1) < debut(c_2) \qquad \forall\, \mathfrak{c} \in \mathfrak{C}, \text{ et } c_1, c_2 \in C^{\mathfrak{c}}, \text{ tels que } c_1 \neq c_2 \text{ et } debut(c_1) \leq debut(c_2) \tag{6.15}$$

Un taxi a besoin de temps pour se déplacer. Si une charge est la première d'un intervalle modifiable d'un taxi, le taxi doit atteindre le chargeur depuis l'origine de cet intervalle

(6.16). Il en va de même pour la dernière charge et la destination de l'intervalle (6.17). Il doit aussi se déplacer entre les chargeurs (6.17). Autrement dit : $\forall\, x \in X, i \in \langle I^x \rangle$:

$$debut(c_1^{x,i}) \geq debut(i) + duree(origine(i), v(c_1^{x,i})) \tag{6.16}$$

$$fin(c_{-1}^{x,i}) \leq fin(i) \quad - duree(v(c_{-1}^{x,i}), destination(i)) \tag{6.17}$$

$$debut(c_{k+1}^{x,i}) \geq fin(c_k^{x,i}) + duree(v(c_k^{x,i}), v(c_{k+1}^{x,i})) \qquad \forall\, k \in \mathbb{N} - \{1, -1\} \tag{6.18}$$

6.3.2 Définition de 3-partitions restreint

Pour montrer que PGC est au moins fortement NP-complet, on montre qu'il existe une réduction polynomiale de 3-partitions (3PR) restreint [Garey et al., 1979], qui est fortement NP-complet, vers PGC. On définit ainsi 3PR.
On appelle multi-ensemble un ensemble dans lequel un élément peut apparaître plusieurs fois.

L'objectif de 3-partitions est de découper un multi-ensemble d'entiers S en m multi-ensembles de 3 éléments dont la somme de chacun vaut une certaine valeur B. Dans sa version restreinte (3PR), les valeurs des éléments de S sont comprises entre $\frac{B}{4}$ et $\frac{B}{2}$. Présentons cela plus formellement.

Données du problème

On a un multi-ensemble d'entiers $S = \{s_1, ..., s_n\}$. Cet ensemble est tel que $\exists!\ m, B \in \mathbb{N}^+$ pour lesquels :

$$|S| = 3m \tag{6.19}$$

$$mB = \sum_{s \in S} s \tag{6.20}$$

$$\frac{B}{4} < s < \frac{B}{2} \qquad \forall\, s \in S \tag{6.21}$$

Solution du problème

Une solution pour 3PR est un multi-ensemble $SOL = \{S_1, ..., S_m\}$ dans lequel chaque élément est un multi-ensemble d'entiers valant B (équation 6.22) et contenant les mêmes éléments que S (équation 6.23) :

$$B = \sum_{s \in S_k} s \qquad \forall\, k \in \mathbb{N}_{[1,m]} \tag{6.22}$$

$$S = \{s \mid S' \in SOL \text{ et } s \in S'\} \tag{6.23}$$

On peut ajouter explicitement comme contrainte que tous les multi-ensembles d'une solution contiennent 3 éléments :

$$|S_k| = 3 \qquad \forall\, k \in \mathbb{N}_{[1,m]} \tag{6.24}$$

Cette contrainte existe dans la version de base de 3-partitions. Elle n'est pas nécessaire dans 3PR, qui limite la valeur des données du problème avec l'équation 6.21. La figure 6.5 présente un exemple de 3PR.

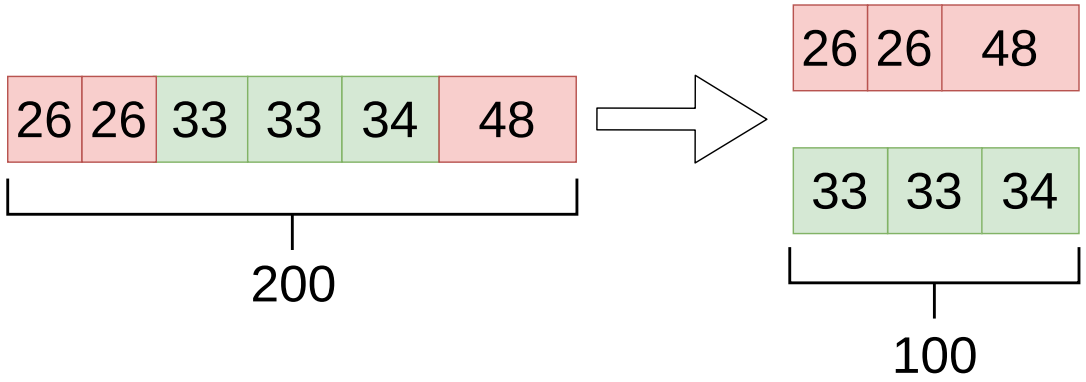


Figure 6.5 – *Un exemple de 3-partition restreint. A gauche, un multi-ensemble d'entiers* $S = \{26, 26, 33, 33, 34, 48\}$*. A droite, un découpage en* $m = 2$ *multi-ensembles de 3 éléments, chacun de valeur totale* $B = 100$*. La valeur de tous les éléments de* S *est dans* $]\frac{B}{4}, \frac{B}{2}[$

6.3.3 Réduction de 3-partitions restreint vers PGC

Montrons maintenant qu'il existe une réduction polynomiale de 3PR vers PGC. Pour cela, nous allons générer une instance de PGC à partir d'une instance de 3PR, et montrer qu'il existe une solution pour cette instance de 3PR si et seulement s'il existe une solution pour l'instance de PGC.
L'idée générale est que dans l'instance de 3PR, pour qu'une solution existe, chaque chargeur doit y être utilisé par exactement 3 taxis (un chargeur représente un ensemble S_k), et chaque taxi ne peut y faire qu'une charge (un taxi représente un élément de s).

Génération d'une instance de PGC à partir d'une instance de 3PR

A partir de ce point, pour distinguer clairement les variables de 3PR et de PGC, les variables de 3PR sont notées avec un accent circonflexe (par exemple, $\hat{B}$).
Une instance de PGC peut être générée à partir d'une instance de 3PR de la manière suivante.

- Dans le graphe $G = (V, A)$, tous les arcs ont une consommation et une durée nulle.
- $t_0 = 0$
- Chaque chargeur $\mathfrak{c} \in \mathfrak{C}$ est tel que $w_{\mathfrak{c}} = 1$.
- Les chargeurs sont toujours disponibles, donc $|\langle Dispo \rangle| = \langle [t_0, t_{fin}] \rangle$.
- Chaque taxi x n'a qu'un seul intervalle modifiable allant de t_0 à t_{fin}. On note $w_{min}(x) = w_{min}^{i_1^x}$ et $w_{max}(x) = w_{max}^{i_1^x}$.
- $manoeuvre$ a une valeur non nulle quelconque.

De plus, $t_{fin} = \hat{B} + 3 \times manoeuvre$, il y a $\hat{m}$ chargeurs. Il y a $|\hat{S}|$ taxis tels que :

$$w_{min}(x_k) = w_{max}(x_k) = \hat{s_k} \qquad \forall\, k \in \mathbb{N}, x_k \in X, \hat{s_k} \in \hat{S} \tag{6.25}$$

Ayant ainsi généré une instance de PGC à partir d'une instance de 3PR, montrons que cette instance de 3PR a une solution si et seulement si l'instance de PGC associée a une solution.

Génération d'une solution de 3PR à partir d'une solution de PGC

Montrons que dans cette instance, s'il existe une solution, alors chaque taxi ne se charge qu'une fois, et chaque chargeur est utilisé trois fois.
Puisque chaque taxi x_k a besoin d'une quantité d'énergie exactement égale à $\hat{s_k}$, que les taxis ne perdent jamais d'énergie durant l'intervalle $[t_0, t_{fin}]$, et que tous les chargeurs ont une puissance de 1 alors :

$$\underbrace{manoeuvre \times |C|}_{\text{Temps total de manoeuvre}} + \underbrace{\sum_{\hat{s} \in \hat{S}} \hat{s}}_{\text{Quantité d'énergie chargée}} = \underbrace{\left(\sum_{c \in C} fin(c) - debut(c) - manoeuvre) \right)}_{\text{Temps total de charge}} \tag{6.26}$$

Notons $usage$ la durée moyenne pendant laquelle un chargeur est utilisée. Puisque $|\mathfrak{C}| = \hat{m}$, alors :

$$usage = \frac{1}{\hat{m}} \left(manoeuvre \times |C| + \sum_{\hat{s} \in \hat{S}} \hat{s} \right) \tag{6.27}$$

Or, $\sum_{\hat{s} \in \hat{S}} \hat{s} = \hat{B}\hat{m}$ donc :

$$usage = \hat{B} + manoeuvre \times \frac{|C|}{\hat{m}} \tag{6.28}$$

Puisque tout taxi doit se charger au moins une fois ($\forall\, x \in X, w_{min}(x) > 0$), on a :

$$|C| \geq |X| = |\hat{S}| = 3\hat{m} \tag{6.29}$$

Donc :

$$usage \geq \hat{B} + manoeuvre \times 3 \tag{6.30}$$

La durée maximale d'utilisation d'un chargeur donné est $t_{fin} - t_0 = \hat{B} + manoeuvre \times 3$. Autrement dit, la durée maximale d'utilisation d'un chargeur est inférieure ou égale à la durée moyenne d'usage d'un chargeur, donc $\forall\, \mathfrak{c} \in \mathfrak{C}$, sa durée d'utilisation $usage_{\mathfrak{c}}$ est telle que :

$$usage_{\mathfrak{c}} = usage = \hat{B} + manoeuvre \times 3 \tag{6.31}$$

On peut donc conclure que dans une solution de cette instance, chaque taxi se charge exactement une fois, et chaque chargeur est utilisé trois fois. Avec cette instance particulière, PGC peut être reformulé de la manière suivante.

On a un multi-ensembles de charges à attribuer à des chargeurs. Chaque charge à une longueur s_k. On appelle S le multi-ensemble contenant la durée de chaque charge. On a un ensemble de m chargeurs.

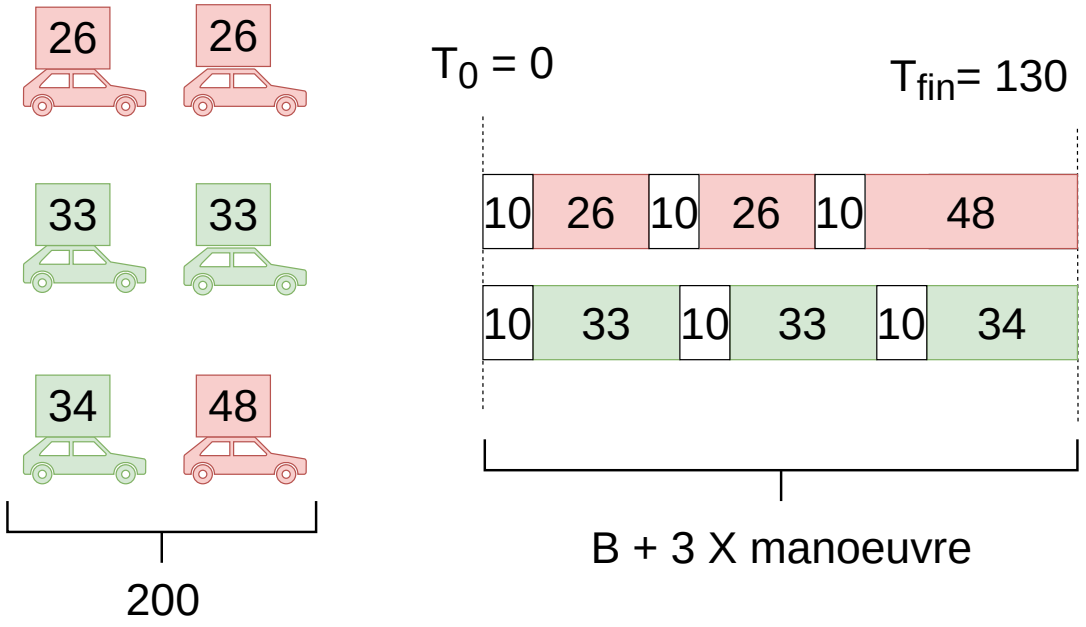


Figure 6.6 – *A gauche, des taxis avec une certaine quantité de batterie à charger. A droite, l'emploi du temps de deux chargeurs. Chaque chargeur gère 3 taxis. La durée des manœuvres est de 10.*

Trouver une solution pour PGC dans cette instance revient à trouver un multi-ensemble $SOL = \{S_1, ..., S_m\}$ tel que, $\forall\ k \in \mathbb{N}_{[1,m]}$, on a :

$$|S_k| = 3 \tag{6.32}$$

$$B = \sum_{s \in S_k} s \tag{6.33}$$

$$S = \{s \mid S' \in SOL \text{ et } s \in S'\} \tag{6.34}$$

Ces trois équations sont exactement les 3 équations qui définissent une solution de 3PR (équations 6.22, 6.23, 6.24). Il suit que si SOL existe, alors SOL est aussi une solution pour 3PR.

On conclut par la même occasion que s'il n'existe pas de solution pour cette instance de PGC, alors il n'existe pas de solution pour l'instance de 3PR correspondante.

C.Q.F.D.

La figure 6.6 présente un exemple de PGC avec ces contraintes, et une solution.

6.4 Présentation de l'algorithme glouton, avec charges

Comme nous l'avons mentionné, la génération de charge à lieu durant l'exécution de la méthode de la section 6.2.4 (Insertion de requête dans un edtt). Après avoir généré un nouvel edtt en insérant une requête dans un edtt existant, il est systématiquement vérifié que les contraintes de batteries sont respectées, et si elles ne sont pas respectées, des charges sont insérées dans l'edtt.

Puisque le sous-problème PGC est au moins NP-complet au sens fort, nous avons développé l'heuristique suivante pour insérer des charges dans un edtt lorsqu'un problème de batterie est détecté.

6.4.1 Insertion de charges dans un intervalle modifiable

Les charges sont insérées dans les intervalles modifiables de l'edtt, qui sont composés de toutes les activités non-course. Chaque charge insérée doit être associée à un chargeur.

Pour rappel, $\alpha(t)$ et $\omega(t)$ dénotent le début et la fin d'un intervalle de temps t quelconque ; $arrive(a, b)$ est l'instant auquel un taxi peut *arriver* à l'origine de b au plus tôt, en partant de la destination de l'activité a à l'instant où a se finit ; et $depart(a, b)$ est l'instant auquel le *départ* de la destination de a doit avoir lieu au plus tard pour arriver à l'origine de b avant le début de b.

Pour insérer des charges à un chargeur $\mathfrak{c}$ durant un intervalle modifiable $\langle I \rangle$, on crée d'abord une charge c à ce chargeur sans initialiser son instant de début et de fin.

On crée ensuite τ tel que qui est l'intervalle de temps entre les trajets à vide nécessaires pour accéder au chargeur durant $\langle I \rangle$. Autrement dit :

$$\alpha(\tau) = arrive(i_0, c) \tag{6.35}$$
$$\omega(\tau) = depart(c, i_{-0}) \tag{6.36}$$

On établit ensuite une liste L des sous-intervalles de temps contenus dans τ durant lesquels le chargeur est disponible, *i.e.* ni utilisé par un autre taxi, ni inutilisable (voir section 4.1.4).

Ayant L, on peut maintenant générer un intervalle modifiable alternatif $\langle I' \rangle$, pour remplacer $\langle I \rangle$. L'intervalle $\langle I' \rangle$ est initialement vide, et on définit $i'_0 = i_0$ et $i'_{-0} = i'_{-0}$.

Pour remplir $\langle I' \rangle$, on parcourt L. Pour chaque intervalle de temps $l \in L$, on ajoute dans $\langle I' \rangle$ une charge au chargeur $\mathfrak{c}$ commençant à l'instant $\alpha(l)$ et finissant à l'instant $\omega(l)$. Après chaque insertion de charge dans $\langle I' \rangle$, on contiguïse $\langle I' \rangle$ (en incluant i'_0 et i'_{-0}).

On obtient à la fin un intervalle modifiable $\langle I' \rangle$ ayant le même début et la même fin que $\langle I \rangle$, et contenant, si possible, des charges au chargeur $\mathfrak{c}$. La figure 6.7 présente une insertion de charges dans un intervalle modifiable.

Une charge n'est insérée que si elle répond à certains critères. Elle ne peut pas charger une quantité trop faible d'énergie (c'est probablement le cas si la charge précédente est une charge complète).

De la même manière, une charge donnant -1% de batterie (tellement courte que le taxi se déconnecte dès qu'il a fini la manœuvre de connexion au chargeur) peut être refusée.

Un paramètre de l'algorithme glouton est un pourcentage minimal de batterie ajouté par charge, exprimé en pourcents de la quantité d'énergie maximale pouvant être contenue dans la batterie du taxi. Un taxi a une batterie considérée *pleine* si et seulement si la quantité d'énergie qui lui manque est trop faible pour qu'une charge soit envisageable.

Dans une solution finale pour le problème statique, une charge peut être arrêtée dès que le niveau de batterie du taxi est de 100%. Dans une solution intermédiaire générée par une heuristique, il peut cependant être intéressant de laisser la charge durer un peu plus, même si le taxi ne gagne effectivement plus d'énergie. Cela permet d'avoir un tampon qui absorbe les changements d'énergie lors des futures modifications de l'edtt. Par exemple, si, après

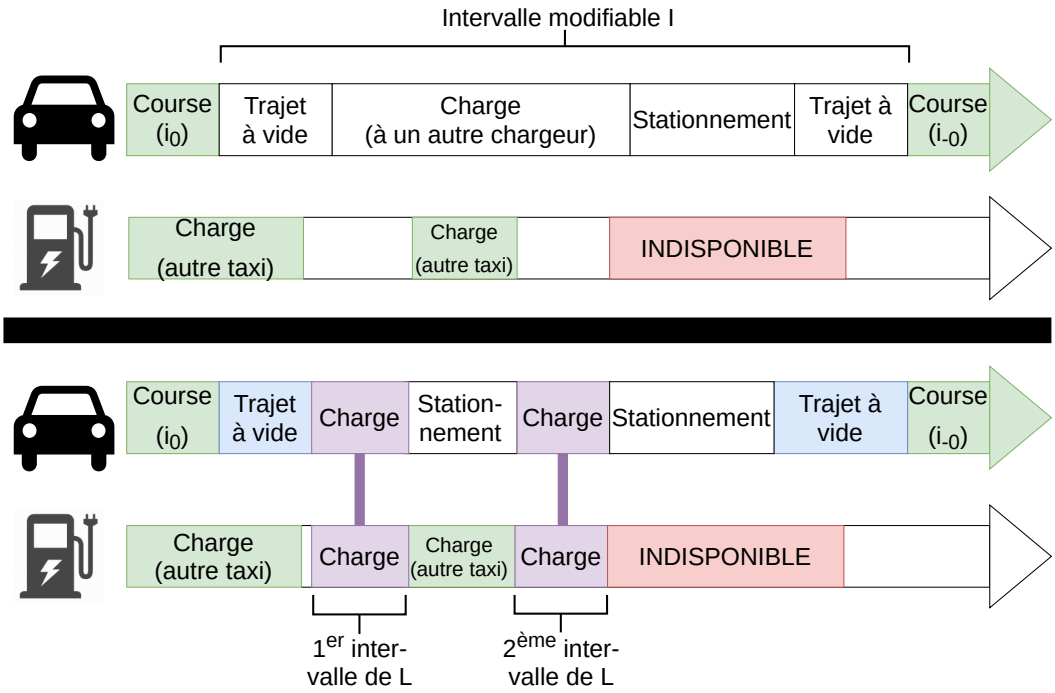


Figure 6.7 – *Un edtt et un edtc, avant (haut) et après (bas) une insertion de charges dans un intervalle modifiable. Les activités en vert sont celles qui ne peuvent pas être modifiées par l'insertion de charges. "INDISPONIBLE" est une fenêtre de temps durant laquelle le chargeur ne peut pas être utilisé ($\notin \langle Dispo \rangle$). Dans cet exemple, une charge à un chargeur peu puissant est remplacée par deux charges plus courtes à un chargeur plus puissant.*

un changement de l'edtt, la consommation d'énergie avant la charge augmente, puisque la charge est légèrement surdimensionnée, le taxi a toujours une batterie complètement pleine à la fin de la charge, sans ajustement supplémentaire nécessaire de son edtt.

Un autre paramètre de l'algorithme glouton est donc une proportion maximale superflue de charge.

L'algorithme 2 décrit en détails l'insertion des charges dans un intervalle modifiable pour un chargeur donné.

Afin d'insérer des charges dans un intervalle modifiable $\langle I \rangle$, on génère d'abord un intervalle $\langle I^{\mathfrak{c}} \rangle$ pour chaque chargeur $\mathfrak{c}$ avec l'algorithme 2. On crée également un intervalle $\langle I^{none} \rangle$ dans lequel le taxi n'utilise aucun chargeur. On sélectionne ensuite parmi ces intervalles celui maximisant le niveau de batterie final du taxi (après le second trajet à vide).

6.4.2 Insertion de charge dans un edtt

Après une insertion de course, on doit vérifier que les contraintes de batterie sont respectées. Premièrement, les taxis ne doivent jamais finir une activité sans énergie. Deuxièmement, les taxis doivent avoir une certaine quantité d'énergie à la fin de leur edtt, et troisièmement, la variation et durée des charges doit respecter les paramètres de l'algorithme glouton (quantité minimale et maximale d'énergie gagnée durant une charge).

On vérifie que ces contraintes sont respectées en parcourant les activités de l'edtt dans l'ordre. Si toutes les contraintes sont respectées, aucune charge n'est ajoutée dans l'edtt.

Supposons maintenant qu'on trouve une course c commençant avec un niveau de batterie négatif. Dans ce cas, on insère des charges dans l'edtt le plus tôt possible, *i.e.* on insère des charges dans chaque intervalle modifiable précédant c, par ordre chronologique.

Après ces insertions, soit c commence toujours avec une batterie négative, et l'insertion de charge renvoie une erreur, soit elle devient valide, et on continue la vérification des contraintes de charges de l'edtt à partir de c.

De la même manière, si le taxi finit avec une quantité d'énergie insuffisante dans cet edtt, des charges sont ajoutées dans les intervalles modifiables de l'edtt.

6.4.3 Insertion de charges dans la solution initiale

Dans la solution initiale, chaque taxi reste stationné. Certains taxis commencent avec un niveau de batterie inférieur au niveau minimal de batterie en fin d'edtt. On insère donc des charges séquentiellement dans l'edtt de chacun de ces taxis. Ces edtt contiennent un unique intervalle modifiable qui va de t_0 à t_{fin}.

Si un taxi se retrouve incapable de se recharger suffisamment après cette insertion de charges (Par exemple, tous les chargeurs sont utilisés par les autres taxis de t_0 à t_{fin}), l'algorithme glouton échoue sans trouver de solution viable. Dans la section suivante, nous montrons que trouver une solution dans une instance sans requêtes est fortement NP-complet dans le cas général.

algorithme 2 – Insertion de charges dans un intervalle modifiable

ENTREE : Un intervalle modifiable $\langle I \rangle$, un chargeur c avec $edtc$ son emploi du temps
SORTIE : Un intervalle modifiable $\langle I' \rangle$

DEBUT :

// On commence par créer la première et la dernière activité de l'intervalle
// modifiable en sortie. Ces extrémités sont les activités bleues de la figure 6.7.
Créer un intervalle modifiable $\langle I' \rangle$ initialement vide
Ajouter dans cet intervalle un trajet à vide t_1 de i_0 vers le chargeur.
Créer un trajet à vide t_2 du chargeur vers i_{-0}, sans l'ajouter à $\langle I' \rangle$

// On liste les intervalles de temps durant lesquelles
// des charges pourront être insérées.
Créer une liste L contenant tous les intervalles de temps $t \subseteq T$ commençant après $fin(t_1)$ et finissant avant $debut(t_2)$ durant lesquels :

- Il n'existe pas de charge $c \in edtc$ dans l'intervalle de temps t (donc $fin(c) \leq \alpha(t)$ ou $debut(c) \geq \omega(t)$) **et**
- Les chargeurs sont utilisables ($\exists\, d \in \langle Dispo \rangle$ tel que $t \subseteq d$) **et**
- t est maximal ($t \not\subset t'$ avec t' respectant les deux autres contraintes)

// On insère autant de charges que possible, et le plus tôt possible.
// Ces charges sont en violet dans la figure 6.7.
pour chaque $t \in L$ **faire**
  ajouter dans $\langle I' \rangle$ une charge c commençant à $\alpha(t)$.
  Contiguïser $\langle I' \rangle$ entre c et l'activité de $\langle I' \rangle$ qui la précède.
  Donner à c la plus longue durée acceptable, avec une fin au plus tard à $\omega(t)$
  si c est trop courte **alors**
    Retirer c de $\langle I' \rangle$
  si La batterie du taxi est pleine à la fin de $\langle I' \rangle$ **alors**
    Sortir de la boucle

// On ajoute les derniers élements de $\langle I' \rangle$
Insérer t_2 à la fin de $\langle I' \rangle$.
Contiguïser $\langle I' \rangle$ entre la dernière et avant-dernière activité de
renvoie $\langle I' \rangle$

6.4.4 Parallélisation

Sans charges, l'algorithme d'insertion d'une requête dans un edtt ne nécessite aucune information concernant ce que font les autres taxis. On peut donc calculer en parallèle pour chaque taxi le résultat de l'insertion de la requête dans son edtt.

Avec charges, il est nécessaire de connaitre les edtc, mais pas comment une insertion de la requête dans un autre edtt influe sur les edtc, donc on peut toujours calculer en parallèle le résultat de l'insertion de la requête dans chaque edtt.

Afin d'accélérer l'exécution de l'algorithme glouton, nous avons utilisé ces propriétés dans notre implémentation.

6.5 Complexité algorithmique de l'algorithme glouton

Ayant présenté l'algorithme glouton dans sa totalité, on peut maintenant parler de la complexité des étapes qui le composent.

Dans cette section, la complexité algorithmique de chaque composant de l'algorithme glouton est évaluée. Présentons d'abord les composants avec une complexité triviale à déterminer. Lorsque ce n'est pas précisé, la complexité algorithmique donnée est à la fois la complexité moyenne et la complexité dans le pire des cas.

6.5.1 Composants ayant une complexité simple à déterminer

Insertion de requête, sans insertion de charges

Contiguïser (section 6.1.2) une séquence de n activités est en $O(n)$. Cette opération peut être accélérée si l'on sait qu'une partie de la séquence est déjà contiguë.

Insérer une course (section 6.2.3) dans un intervalle modifiable i est en $O(i)$. Le cardinal d'un edtt est, en ordre de grandeur, proportionnel au nombre de courses qu'il contient. Insérer une requête dans un edtt e (section 6.2.4) se fait donc en $O(|e|\alpha)$ avec α la complexité de l'insertion d'une requête dans une course pour former une course partagée. Nous verrons dans la suite de cette section que :

— $\alpha = O(k^3)$ avec k le nombre de stops de la course dans le cas général.

— $\alpha = O(k^2)$ si la congestion est constante.

— $\alpha = O(1)$ ou $O(k)$ dans un grand nombre de cas

Soit ϵ la somme des cardinaux des edtt d'une solution. Une itération de l'algorithme glouton est en $O(\epsilon\alpha)$. Pour calculer la complexité de l'algorithme glouton dans sa globalité, on va distinguer deux cas : avec et sans partage de course.

Sans partage, $\alpha = O(1)$, et à l'itération k, on a $\epsilon \approx k+|X|$. Donc l'insertion de l'intégralité de Q dans une solution initialement vide a une complexité $O(|Q|(|Q| + |X|))$. En détaillant plus le calcul, le nombre d'opération est environ $|X|+(|X|+1)+...+(|X|+|Q|) = |Q||X|+\frac{|Q|(|Q|+1)}{2}$.

Avec partage, le pire cas de figure est atteint lorsque toutes les requêtes sont insérées dans la même course partagée. Dans ce cas, la complexité de l'insertion à l'itération k de l'algorithme glouton est $O(k^3 + |X|)$, et la complexité globale de l'algorithme est $O(|Q|^4 + |Q||X|)$.

En règle générale, $|Q| \gg |X|$, donc la complexité effective est plutôt entre $O(|Q|^2)$ et $O(|Q|^4)$, en fonction du taux de partage.

Insertion de charges

Une insertion de charges à un chargeur donné dans un intervalle modifiable (algorithme 2) requiert de parcourir $\langle Dispo \rangle$ (section 4.1.4) et l'edtc e du chargeur, afin de lister tous les intervalles de temps durant lequel le chargeur est utilisable durant cet intervalle de temps. Cela peut être fait simplement en $O(|Dispo| + |e|)$.

Pour rappel, $\mathfrak{C}$ est l'ensemble des chargeurs. Une insertion de charges dans un intervalle modifiable est fait en $O(|\mathfrak{C}||Dispo| + |C_{utilise}|)$ avec $C_{utilise} \subseteq C$ l'ensemble des sommets de charges parcourus par l'ensemble des edtt de la solution dans le graphe des activités.

Avec Q^x l'ensemble des requêtes traitées par le taxi $x \in X$, et puisque un edtt e contient au plus $Q^x + 1$ intervalles modifiables, une insertion de charges dans un edtt (section 6.4.2) est faite en :

$$O(|Q^x|(|\mathfrak{C}||Dispo| + |C_{utilise}|)) \tag{6.37}$$

Cette opération peut être accélérée, car les calculs effectués durant les insertions dans un intervalle peuvent être utilisés pour l'intervalle suivant. Supposons par exemple que $dispo_k$ est le dernier intervalle de $\langle Dispo \rangle$ dans un intervalle modifiable. Dans ce cas, on sait que le premier intervalle de $\langle Dispo \rangle$ dans l'intervalle modifiable suivant est $dispo_{k+j}$ avec $j \geq 0$. Ainsi, une insertion de charges dans un edtt peut être faite en :

$$O(|Q^x| + |\mathfrak{C}||Dispo| + |C_{utilise}|) \tag{6.38}$$

La complexité de l'itération k de l'algorithme glouton avec charge et sans partage est, en supposant que $C_{utilise}$ est proportionnel à k, de l'ordre de $O(|X|(k+|\mathfrak{C}||Dispo|))$. La complexité globale de l'algorithme glouton avec charge et sans partage est de $O(|Q||X|(|\mathfrak{C}||Dispo| +|Q|))$.

Concernant le glouton avec partage et avec charge, puisque l'insertion de charge est faite de manière séquentielle, la complexité globale est $O(|Q|^4 + |QX|(|\mathfrak{C}||Dispo| + |Q|))$.

En règle générale, $|Q|$ est supérieur à $|X|, |\mathfrak{C}|$ et $|Dispo|$, donc on a une complexité en $O(|Q|^4)$ avec partage. Sans partage, avec $|Q| > |X|$ et $|Q|^2 > |X||\mathfrak{C}||Dispo|$, on a une complexité générale de $O(|Q|^3)$.

6.5.2 Complexité non-triviale, Création de course partagée

La création d'une course partagée a une complexité élevée sans optimisation, mais elle peut être diminuée à l'aide de pré-calculs.

Rappelons quelques notations qui vont être utilisées dans ce chapitre. $\langle P^c \rangle$ est la suite des portions de la course c, et une portion est associé à un arrêt (montée ou descente de client). Dans cette section :

- $arrive(c, k)$ dénote l'instant auquel commence l'arrêt de la portion k de la course c.
- $depart(c, k)$ dénote l'instant auquel se finit la portion k de la course c.
- $\omega(c, k)$ est le dernier instant auquel l'arrêt peut avoir lieu.
- passager(c, k) est le nombre de passagers dans le taxi avant l'arrêt.

Recherche exhaustive

Le nombre d'emplacements possibles pour s_q^+ et s_q^- est $\binom{2}{|P^c|+1} + |P^c| - 1$. Tester une combinaison se fait en $O(|P^c|)$, car on teste pour chaque stop si ses contraintes de temps et de capacité sont respectées. La complexité totale de l'insertion est donc, en utilisant une recherche exhaustive $O(n^3)$. C'est cette méthode qui a été présentée plus tôt dans ce chapitre. Dans une instance dans laquelle la congestion est constante, on peut trouver un ordre des stops valide (respectant toutes les contraintes de temps et de capacité) en $O(n^2)$, si un tel ordre existe.

On appelle (i, j) la configuration de $\langle P^{c'} \rangle$ dans laquelle la montée de q est dans la i-ème portion de c', et la descente de q est dans la j-ème portion.

Amélioration, sans action de stationnement en course.

Dans cette sous-section, on suppose qu'un taxi qui arrive à l'arrêt d'un client le récupère immédiatement et repars immédiatement. Il n'y a donc pas d'action de stationnement durant une course. Cette hypothèse est relaxée dans la sous-section suivante.

Pour obtenir une complexité en $O(n^2)$, on teste $O(n^2)$ configurations de $P^{c'}$ et le test de chaque configuration se fait en $O(1)$ en moyenne, grâce à un ensemble de pré-calculs. Pour déterminer si la configuration (i, j) est valide, il faut s'assurer que :

I $depart(c', i) \leq \omega(c', i)$

II $depart(c', j) \leq \omega(c', j)$

III $depart(c', k) \leq \omega(c', k)\ \forall\ k \in \mathbb{N}_{]i,j[}$

IV $depart(c', k) \leq \omega(c', k)\ \forall\ k \in \mathbb{N}$ tel que $k > j$

V $passagers(c', k) \leq places_{taxi(c)}\ \forall\ k \in \mathbb{N}$ tel que $i \leq k \leq j$

La première étape pour tester la configuration (i, j) consiste à calculer $depart(c', i)$. Si $i = 1$, on peut le calculer directement à partir de $depart(c, 1)$. Sinon, $depart(c', i)$ peut être calculé directement à partir de $depart(c, i-1)$. La condition I peut donc être vérifiée en temps constant.

Pour vérifier les conditions II, III, et IV en temps constant, leur vérification est faite sans calculer la valeur des autres $depart$. Pour cela, on définit la notion de retard. Le retard $r(k)$ est la différence entre l'instant de départ de la portion k dans c' et dans la portion correspondante dans c. Autrement dit :

$$r(k) \text{ est indéfini pour} \quad k = i \text{ et } k = j \tag{6.39}$$

$$r(k) = 0 \quad \forall\ k < i \tag{6.40}$$

$$r(k) = depart(c', k) - depart(c, k-1) \quad \forall\ i < k < j \tag{6.41}$$

$$r(k) = depart(c', k) - depart(c, k-2) \quad \forall\ j < k \tag{6.42}$$

On suppose que le taxi quitte toujours un stop dès qu'il y arrive. Dans ce cas, puisqu'il n'y a pas de congestion, le retard est parfaitement transitif, autrement dit, tout retard subi à un stop k est retransmis au stop $k + 1$. Le retard au stop $j - 1$ est donc égal au retard subi au stop $i + 1$. La condition II peut donc être vérifiée en temps constant, en calculant le retard au stop i.

Vérifier les conditions III et IV en temps constant demande des pré-calculs. Notons $rt(k)$ le retard toléré au stop k dans c. On a :

$$rt(k) = \omega(c,k) - depart(c,k) \tag{6.43}$$

Pré-calculer les rt est fait en $O(P^c)$. Vérifier la condition III revient à assurer que :

$$r(k) \leq rt(k-1) \qquad \forall\, k \in \mathbb{N}_{]i,j[} \tag{6.44}$$

Puisque $r(k+1) = r(k)$ dans cet intervalle, cette condition est validée si et seulement si :

$$r(i+1) \leq \min_{i<k<j} rt(k-1) \tag{6.45}$$

De la même manière, pour la condition IV :

$$r(j+1) \leq \min_{j<k} rt(k-2) \tag{6.46}$$

On peut noter que, puisque $r(i+1) \leq r(j+1)$, alors vérifier les conditions III et IV revient à vérifier que :

$$r(i+1) \leq \min_{i<k} rt(k-1) \tag{6.47}$$

$$r(j+1) \leq \min_{j<k} rt(k-2) \tag{6.48}$$

Pour vérifier en temps constant que ces deux équations sont vérifiées, il suffit de pré-calculer $\min_{k\in\mathbb{N}} rt(k)$. Enfin, la condition V peut être validée en temps constant pour la configuration $(i, i+1)$. Si elle n'est pas validée par la configuration (i,j), alors elle n'est pas validée par la configuration $(i, j+1)$. Sinon, il suffit de vérifier que $passagers(c', j+1) \leq places_{taxi(c)}$, pour savoir si la configuration $(i, j+1)$ valide la condition V.

En résumé, les pré-calculs sont en temps linéaire, on teste $O(|P^c|^2)$ combinaisons, et la validité de chaque combinaison peut être vérifiée en temps constant. L'insertion de requête dans une course partagée peut donc être fait en $O(|P^c|^2)$, à condition que la congestion soit constante et que $\forall\, s$, on ait $arrive(c,s) = depart(c,s)$. On peut aussi noter que, souvent, il n'est pas nécessaire de tester toutes les configurations, car si une condition n'est pas validée par la configuration (i,j), elle peut être impossible à valider par la configuration $(i, j+k)$ avec $k \geq 0$. Les performances peuvent donc souvent être meilleures que $O(|P^c|^2)$.

Amélioration, avec action de stationnement

Considérons maintenant le cas où il existe k tel que $arrive(c,k) \neq depart(c,k)$. Dans ce cas, une partie du retard peut être absorbée.

Appelons $t(k)$ le tampon de la portion k d'une course c tel que :

$$t(k) = depart(c,k) - arrive(c,k) \qquad \forall\, k \in \mathbb{N} \tag{6.49}$$

Notons $t(a,b)$ la somme des tampons entre l'arrêt a et l'arrêt b. Autrement dit :

$$tampon(a,b) = \sum_{z=a}^{b} tampon(z) \tag{6.50}$$

Le *retard effectif subi* est le retard qui n'est pas absorbé par les tampons. Dans la configuration (i, j), le retard effectif subi au sommet $s_k^{c'}$ est noté $res(k)$, tel que :

$$res(k) = max(0, r(i+1) - tampon(i, k-1)) \qquad \forall\, k \in \mathbb{N}_{]i,j]} \tag{6.51}$$

$$res(k) = res(j) + max(0, r(j+1) - tampon(j+1, k)) \qquad \forall\, k \in \mathbb{N}, k > j \tag{6.52}$$

Calculer $res(k)$ peut être fait en temps constant pour tout k. Pour déterminer si les conditions III et IV sont validées :

$$r(i+1) \leq \min_{i<k} rt(k-1) \text{ devient } res(i+1) \leq \min_{i<k} rt(k-1) \tag{6.53}$$

$$r(j+1) \leq \min_{j<k} rt(k-2) \text{ devient } res(j+1) \leq \min_{j<k} rt(k-2) \tag{6.54}$$

Tous les tests sont effectués en temps constant, tous les pré-calculs au plus en temps linéaire de $|P^c|$. L'insertion de requête dans une course partagée peut donc être fait en $O(|P^c|^2)$, à condition que la congestion soit constante.

Si la congestion est variable, alors le retard n'est pas transitif (on peut avoir $r(k+1) \neq r(k)$). Dans ce cas, il est impossible de déterminer $depart(c', -1)$ dans la configuration (i, j) à partir de sa valeur dans une autre configuration. Les conditions I, II, III et V peuvent être effectuées en temps constant en réutilisant les valeurs calculées pour d'autres configurations, mais la condition IV force de recalculer $depart(c', k) \forall\ k > j$ dans chaque configuration, avec une complexité en $O(P^c)$ pour chaque configuration.

Dans nos instances, pour faciliter diverses comparaisons entre instances avec congestion fixe et avec congestion variable, nous avons choisi de toujours utiliser la recherche exhaustive, y compris lorsque la congestion est fixe.

Chapitre 7

Recuit Simulé

Pour améliorer les résultats trouvés par l'algorithme glouton, dont la principale faiblesse est de ne jamais retirer de requête d'un edtt, un recuit simulé est utilisé. Il s'agit d'une méta-heuristique relativement ancienne [Kirkpatrick et al., 1983] et largement étudiée [Nikolaev and Jacobson, 2010].

7.1 Méta-heuristique par recherche de voisinage

Un recuit simulé est une méta-heuristique par recherche de voisinage. Avant de décrire ce qu'est une telle méta-heuristique, voici quelques définitions.

7.1.1 Définitions

Une **fonction de mouvement** désigne une fonction prenant en paramètre une solution pour le PCE-ADARP, et dont l'unique sortie est une solution pour le PCE-ADARP. Un **mouvement** désigne l'application d'une fonction de mouvement avec une solution donnée en paramètre.

Soit a une solution et m une fonction de mouvement. Le **voisinage** de a induit par m est l'ensemble des solutions qui peuvent être obtenues avec m en ayant a pour paramètre principal.

Une manière de visualiser un ensemble de solutions et les relations entre les solutions est par un **graphe des voisinages**. Dans ce graphe, chaque sommet représente une solution, et chaque arc (u, v) représente un mouvement avec pour paramètre la solution de u et pour sortie la solution de v.

Prenons par exemple le TSP (*cf.* page 11). Une solution du TSP est une suite qui contient chaque sommet du graphe exactement une fois. Un mouvement peut consister en une simple permutation dans l'ordre de passage :

— Paramètres : Une solution du TSP $\langle V \rangle$ qui est une suite de sommets, un indice i et un indice j.

— Sortie : Une solution $\langle V' \rangle$ qui est une copie de $\langle V \rangle$, mais dans laquelle v_i et v_j ont été intervertis.

Pour une solution donnée, il existe $\binom{2}{n}$ permutations, avec n le nombre de sommets du graphe. Chaque solution a donc $\binom{2}{n}$ voisins. Le graphe des voisinages de cette permutation est non-orienté, car toute permutation peut être inversée (si a est voisin de b alors b est voisin de a). De plus, ce graphe des voisinages est connexe, car il n'existe pas de solution qui ne puisse être obtenue avec une suite de permutations depuis n'importe quelle solution.

7.1.2 Fonctionnement

Une recherche de voisinage consiste à :

1. établir une solution initiale, à l'aide d'une heuristique quelconque
2. la modifier incrémentalement à l'aide de mouvements (se déplacer dans le graphe des voisinages correspondant)
3. atteindre un critère d'arrêt
4. renvoyer la meilleure solution trouvée, *i.e.* celle maximisant la valeur de la fonction objectif.

Une telle méta-heuristique a donc quatre composants principaux. Une solution initiale, un ensemble de fonctions de mouvement, un critère déterminant quel mouvement choisir à chaque itération, et enfin, un critère déterminant quand la recherche de solution s'arrête.

Trouver un voisin

Le graphe des voisinages n'est pas toujours généré dans son intégralité durant l'exécution d'une recherche de voisinage, car sa taille ne le permet souvent pas. Il faut cependant pouvoir à chaque itération trouver un voisin de la solution actuelle. Il est donc courant de générer le voisinage de la solution actuelle après chaque mouvement. D'autres stratégies existent pour réduire au maximum la partie générée du graphe des voisinages.

Par exemple, pour le TSP, n'importe quelle permutation (i, j) avec $0 < i < j \leq n$ peut être utilisée pour un mouvement de permutation. Pour choisir un voisin, une méthode peut donc consister à choisir une valeur aléatoire pour i et j, et il n'y a pas besoin de lister les $\binom{2}{n}$ voisins d'une solution, ce qui aurait un impact majeur sur le temps de calcul. Une recherche de voisinage doit donc inclure une stratégie pour trouver le voisinage de chaque solution visitée ou une partie de ce voisinage.

Certains mouvements peuvent avoir pour sortie une solution inadmissible. Ces mouvements peuvent être acceptés, dans l'espoir que passer par une solution inadmissible va permettre d'atteindre de bonnes solutions admissibles. Ils peuvent aussi être refusés, et une solution non-viable n'est jamais considérée comme une voisine. Un paramètre supplémentaire des recherches de voisinage est donc, si besoin, une règle déterminant comment sont traitée les solutions inadmissibles.

Choisir un voisin, et le critère d'arrêt

Il existe une multitude de stratégies pour choisir l'un des voisins d'une solution. Présentons ici deux stratégies basiques : la montée de colline et la marche aléatoire.

Dans une montée de colline [Skiena, 2008], la valeur des mouvements est utilisée pour en choisir un. La valeur d'un mouvement est la différence entre la valeur de la fonction objective pour la solution en entrée et celle en sortie. Dans un problème de maximisation, la valeur d'un mouvement améliorant est positive, et celle d'un mouvement dégradant est négative. Dans une montée de colline, seuls les mouvements améliorants sont acceptés. Il existe deux variantes principales de la montée de colline.

Dans la montée de colline *steepest ascent* [Skiena, 2008], le mouvement de plus forte valeur est choisi à chaque itération. Dans la montée de colline stochastique, n'importe quel mouvement améliorant est choisi. La montée *steepest ascent* n'est possible que lorsqu'il est envisageable de recenser tous les voisins de chaque sommet visité, et de calculer la valeur des mouvements associés. Dans la montée stochastique, un mouvement est généralement tiré en suivant une distribution aléatoire uniforme, et si le mouvement choisi est améliorant, il est accepté, sinon, il est refusé et un nouveau tirage à lieu. Puisqu'une montée de colline n'accepte jamais de mouvement dégradant la solution, la recherche de solution s'arrête au plus tard au moment où une solution n'ayant pas de mouvement améliorant est trouvée. Cette solution est un *optimum local* dans le graphe des voisinages.

Dans une marche aléatoire, chaque voisin a la même probabilité d'être choisi comme prochaine solution. Cette approche à l'avantage de permettre de quitter les solutions localement optimales (n'ayant pas de voisin de plus forte valeur mais pas nécessairement optimale), mais peut prendre beaucoup de temps avant de trouver une solution meilleure que l'initiale. Le critère d'arrêt d'une marche aléatoire peut être le temps d'exécution depuis le début de la marche, ou bien une certaine durée passée sans trouver d'amélioration de la solution, entre autres critères.

7.2 Recuit simulé, cas général

Comme il a été mentionné, le recuit est une recherche de voisinage qui s'apparente à une montée de colline stochastique. Cependant, plutôt que de systématiquement refuser un mouvement dégradant lorsqu'il est tiré, celui-ci peut être accepté avec une certaine probabilité, qui dépend de l'ampleur de la baisse de qualité de la solution, et qui diminue progressivement au cours de l'exécution du recuit simulé.

Au début de son exécution, la recherche de solution a donc le même comportement qu'une marche aléatoire, *i.e.* tout mouvement à la même probabilité d'être choisi, et finit par avoir le même comportement qu'une montée de colline stochastique, avec une transition progressive entre ces deux stratégies.

Cette méthode est inspirée du phénomène thermodynamique de cristallisation et de recuit métallurgique [website, 13]. L'annexe B présente ce principe, et fournit à la fois une intuition sur le fonctionnement du recuit simulé, et la raison pour laquelle cette méta-heuristique peut fournir de bons résultats. L'idée principale est qu'un paramètre du recuit est une variable qu'on appelle la *température* et qui influence la probabilité d'acceptation d'un mouvement. La température est initialement élevée (forte probabilité d'acceptation), et finit proche de 0 (aucun changement dégradant n'est accepté). Toutes choses étant égales par ailleurs, plus la température diminue lentement, plus la solution finale obtenue est bonne. Le plus souvent, la température a une descente par palier, autrement dit, elle reste fixe pendant une certaine durée, puis diminue sensiblement.

Voici une liste exhaustive des paramètres d'un recuit simulé.

— Tous les paramètres communs à toutes les recherches de voisinage mentionnés dans la section précédente.
— Une température initiale.
— Un critère déterminant quand changer de palier de température.
— Une fonction déterminant l'évolution de la température entre chaque palier.
— Une fonction d'acceptation, qui prend pour paramètre un mouvement et une température, et renvoie une probabilité.

Probabilité d'acceptation d'un mouvement

Si un mouvement est améliorant, il est systématiquement accepté. Sinon, il est accepté avec une probabilité p.

Plusieurs formules ont été proposées dans la littérature pour déterminer comment cette probabilité p est calculée. La formule la plus couramment utilisée est celle proposée originellement par [Kirkpatrick et al., 1983]. Elle est inspirée de l'algorithme de Metropolis-Hasting [Hastings, 1970] et de la relation entre recherche de voisinage et calcul de distribution stationnaire dans une chaîne de Markov en temps discret. Cette formule est, avec $\Delta \in \mathbb{R}^-$ la variation de qualité de la solution (négative puisque dégradante dans le cadre d'un problème de maximisation) et $temperature \in \mathbb{R}^+$:

$$p = e^{\frac{\Delta}{temperature}} \tag{7.1}$$

[Dueck and Scheuer, 1990] ont proposé une approche déterministe, dans laquelle une solution dégradante est toujours acceptée lorsque $|\Delta| < temperature$. Cette approche peut se montrer efficace dans certains problèmes, comme le montre [Franz et al., 2001].

Température initiale

La température initiale est généralement déterminée empiriquement, en fonction des caractéristiques du problème, et de sorte que même les mouvements les plus dégradants ont une probabilité d'acceptation proche de 1.

Fonction d'évolution de la température

La variation de température à chaque mise-à-jour est régulière dans la plupart des articles.

[Kirkpatrick et al., 1983] a proposé une évolution géométrique de la température :

$$T_k = T_0 \alpha^k \tag{7.2}$$

où T_0 est la température initiale, T_k, est la température après la k-ème mise-à-jour, et où $0.8 \leq \alpha \leq 0.95$. Il a été prouvé par la suite que le recuit converge vers une solution optimale asymptotiquement, amenant [Alex et al., 1989] à utiliser la formule suivante :

$$T_k = \frac{T_0}{1 + \alpha log(1+k)} \tag{7.3}$$

Cette formule est cependant réputée pour converger trop lentement vers une solution, notamment avec l'analyse de la baisse de température requise pour que la convergence ait effectivement lieu [Granville et al., 1994].

Cela a amené un grand nombre d'auteurs à ignorer cette propriété de convergence, comme l'explique [Robini and Reissman, 2013]. La plupart du temps, lorsqu'un recuit simulé est utilisé, il n'est pas attendu par l'utilisateur que la solution obtenue soit optimale, mais simplement qu'elle soit la meilleure possible pour une durée d'exécution donnée. Le site [website, 7] présente une revue non-exhaustive de formules ayant été testées. Les formules privilégiées utilisent une fonction de température strictement décroissante, mais d'autres formules existent.

[Locatelli, 2000] permet par exemple de réchauffer le système lorsque la solution actuelle est bien plus mauvaise que la meilleure trouvée jusque-là. Cette technique permet de compenser une baisse de température trop rapide.

[Mohamed and Thomas, 2014] réinitialise la température à intervalles réguliers. Dès que la température descend en dessous de 1, elle reprend sa valeur initiale. Réchauffer ainsi le système permet de sortir d'extrema locaux profonds, *i.e.* des solutions ou ensembles de solutions qu'il est difficile de quitter ; le terme bassin profond est parfois utilisé dans le contexte d'un problème de minimisation.

Dans certains cas, une température fixe non-nulle est compétitive [Fielding, 2000,Alrefaei and Andradóttir, 1999].

Changement de palier de température

A temps total d'exécution égal, un arbitrage doit être fait entre nombre de paliers de température et nombre de mouvements par palier. Par exemple, est-il préférable de réduire la température de 10% tous les 10000 mouvements ou de la réduire de 1% tous les 1000 mouvements ?

Il ne semble pas que les recuits avec une réduction majeure (>20%) de la température entre deux paliers soient populaires. Lorsque la variation de température entre deux paliers est faible, [Cohn and Fielding, 1999] note que la régularité de la baisse de température n'a pas d'effet sensible sur la qualité d'un recuit simulé. Par exemple, multiplier la température par 0.9999 après chaque mouvement ne semble pas fournir de résultats différents d'une multiplication de la température par $0.9999^{1000} \approx 0.9$ tous les 1000 mouvements. A notre connaissance, il n'existe pas d'article indiquant qu'une baisse de la température par palier est préférable.

Avoir des changements de température par palier semble surtout avoir pour intérêt de faciliter des preuves de convergence, ou de simplifier l'implémentation du recuit, en particulier lorsque la baisse de la température est adaptative (Adaptive Simulated Annealing). Dans [Kirkpatrick et al., 1983] par exemple, la température est réduite soit au bout d'un certain nombre de mouvement acceptés, soit au bout d'un certain nombre de mouvements rejetés. Avec cette approche, le recuit reste plus longtemps dans des températures modérées, dans lesquels la proportion acceptation/rejet de mouvement est proche d'une valeur déterminée empiriquement.

Avec une baisse de la température non-adaptative, il ne semble pas gênant d'avoir un changement de température entre chaque mouvement, et ainsi avoir une variation continue de la température.

Critère d'arrêt

Il existe, pour résumer, deux critères d'arrêts (qui ont chacun de nombreuses variantes). Le premier est une simple limite sur la durée d'exécution du recuit ou du nombre de mouvements permis.

Le second est utilisé par [Kirkpatrick et al., 1983]. L'algorithme termine si la température change trois fois sans qu'un seul mouvement soit accepté. La température est visiblement trop basse pour que des mouvements soient acceptés, donc le système est considéré *gelé*.

7.3 Recuit appliqué au PCE-ADARP statique

Avant de détailler les fonctions de mouvement utilisées dans notre recuit simulé, détaillons l'évolution et l'impact de la température.

7.3.1 Température

La température initiale a été choisie empiriquement. Sa valeur est précisée dans le chapitre 9.

Nous avons choisi d'avoir une modification faible et continue de la température, plutôt qu'une baisse de la température par palier, ce qui n'a pas d'effet négatif notable, comme le note [Cohn and Fielding, 1999].

Concernant l'évolution de la température, il a été choisi de la faire évoluer en fonction du temps, plutôt qu'en fonction d'un nombre de mouvements. La raison principale est que les mouvements implémentés ont des complexités algorithmiques différentes, et donc que pour les comparer, il est plus judicieux de le faire avec une durée d'exécution fixe. La condition d'arrêt est donc une limite sur la durée d'exécution du recuit.

Quatre règles de mise-à-jour de la température ont été testées. La méthode la plus simple est une descente linéaire. Supposons qu'on souhaite exécuter un recuit durant M minutes, la température initiale est $k_0 \in \mathbb{R}^+$ et la température finale est $k_{fin} \in \mathbb{R}^+$. La température k_m à la minute $m \in \mathbb{R}^+$ est :

$$k_m = k_0 \frac{M-m}{M} + m_M \frac{m}{M} \tag{7.4}$$

La deuxième méthode testée est une réduction géométrique (similaire à [Kirkpatrick et al., 1983]).

La troisième est une décroissance linéaire comme la première, mais la température est réinitialisée à k_0 à intervalle régulier (proche de [Mohamed and Thomas, 2014]).

La dernière méthode est un sinusoïde décroissant, qui utilise la formule suivante. Soit $p \in \mathbb{N}$ le nombre de mouvements nécessaires pour une période :

$$k_m = k_M + \frac{m}{M} \frac{(k_M - k_0)}{2} \times \left(1 + cos\left(\frac{m}{p}\right)\right) \tag{7.5}$$

La figure 7.1 présente la courbe correspondant à cette méthode.

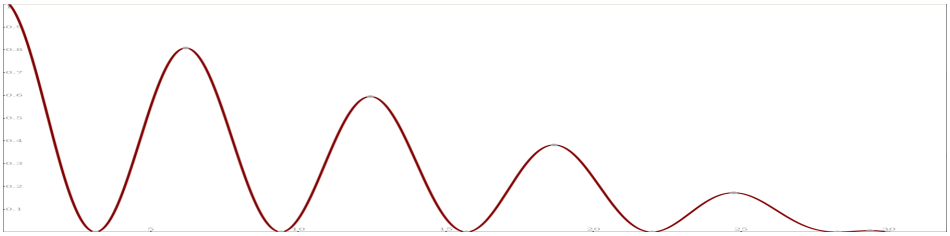


Figure 7.1 – *Variation de température (ordonnée) lors d'une baisse de la température en sinusoïde décroissant, en fonction du temps (abscisse).*

7.4 Mouvements

Dans le PCE-ADARP, un bon mouvement doit permettre 1) l'ajout et la suppression de requêtes traitées, 2) l'envoi d'une requête d'un edtt vers un autre, 3) la modification de l'ordre des montées et descentes des clients au sein d'un edtt, 4) l'ajout et la suppression de charges.

Deux approches permettent d'obtenir ce résultat. 1) Utiliser une fonction de mouvement avec une forte complexité en temps, effectuant toutes ces modifications. 2) Utiliser des fonctions de mouvement avec une plus faible complexité, ce qui permet d'effectuer plus de mouvements, à durée d'exécution du recuit égale. Il faut dans ce second cas attribuer à chaque fonction de mouvement une probabilité d'être tirée à chaque itération.

Dans cette section, des mouvements élémentaires sont présentés, puis des mouvements composites utilisant les mouvements élémentaires sont présentés.

Un mouvement qui génère une solution inadmissible est systématiquement annulé, on ne passe donc jamais durant l'exécution du recuit par une solution inadmissible. La raison principale est qu'il est incertain qu'une recherche de solution qui passe par une solution inadmissible arrive à nouveau à une solution admissible en un temps raisonnable.

7.4.1 Mouvements élémentaires

On utilise deux mouvements simples dans le recuit simulé, l'ajout et le retrait de requête. Un retrait d'activité d'un edtt signifie retirer cette activité, puis contiguïser l'edtt avec la méthode présentée dans la section 6.1.2. Dans cette section, aléatoire signifie "suivant une distribution uniforme".

Ajout de requête (R^+) Ce mouvement a pour paramètre un edtt aléatoire e et une requête non-traitée aléatoire q. Il consiste à ajouter q dans e avec la même méthode que celle de l'algorithme glouton.

Retrait de requête (R^-) Ce mouvement a pour paramètre une requête q traitée dans un edtt e. Dans ce mouvement, la course c qui traite q est retirée de e, puis e est contiguïsé (*cf.* section 6.1.2 pour une définition de contiguïsation). Si c est une course partagée, alors plutôt que de l'enlever, elle est remplacée par une course c' qui est une copie ne contenant pas les gestions de stop de q. Dans c', l'instant auquel le premier client est récupéré est l'instant auquel ce même client est récupéré dans c (donc $debut(c') > debut(c)$ si le premier client

récupéré dans c est q). Concernant les stops suivants, les clients sont récupérés et déposés le plus tôt possible.

7.4.2 Mouvements composés

A partir de ces deux mouvements élémentaires, des mouvements plus complexes peuvent être générés.

Ajout de requête avec retrait (R^{+-}) Ce mouvement commence avec une exécution de R^{+} avec un edtt e aléatoire et une requête non-traitée q aléatoire. Si R^{+} échoue à cause d'une contrainte de batterie violée, le mouvement échoue. Sinon, c'est que R^{+} a échoué à cause d'une contrainte de temps ou de capacité. Dans ce cas, on tolère des retraits de requête, qu'on retire de la manière suivante. On génère une course individuelle c traitant q. L'instant de début de c est celui qui respecte les contraintes de temps de q tout en minimisant la valeur des courses de e incompatibles avec c. On insère c dans e. Les activités incompatibles avec c sont retirées et e est contiguïsé.

Ajout de requête avec glouton ($R^{glouton}$) Ce mouvement commence par une exécution de R^{+-}. Après que des requêtes aient été retirées de l'edtt, elles sont réinsérées dans la solution en utilisant l'algorithme glouton.

Échange de requêtes (R^{swap}) Cette méthode a pour paramètre un edtt e aléatoire, un edtt e' aléatoire, et une requête q aléatoire traitée par e. Cette méthode consiste à trouver la première requête q' dans e' dont le traitement commence après celui de q dans e, puis à échanger ces deux requêtes. Plus exactement, à retirer ces requêtes de leur edtt respectif avec le mouvement R^{-}, puis à insérer q' dans e et q dans e' avec R^{+}.

7.4.3 Charge

Dans le recuit simulé, la méthode d'insertion de charges de l'algorithme glouton est systématiquement utilisée après un mouvement, sauf s'il s'agit d'un mouvement élémentaire utilisé dans le cadre d'un mouvement composé.

Cette nécessité d'ajouter des charges est évidente dans le cas d'un ajout de course, mais on peut aussi noter que même la suppression d'une course peut causer un problème de batterie. En effet, en faisant un détour, un taxi peut réduire sa consommation énergétique, car le plus court chemin n'est pas nécessairement celui qui consomme le moins, en particulier lorsque la congestion varie beaucoup (*cf.* section 3.1.1 pour un rappel sur la manière dont elle est implémentée).

Ayant ainsi défini les caractéristiques des différents mouvements, on peut maintenant présenter les effets de l'utilisation successive de ces mouvements.

7.5 Analyse de la structure de voisinage

Montrons que le graphe des voisinages induit par les mouvements décrits dans ce chapitre n'est pas connexe. En particulier, montrons qu'il n'existe pas toujours un chemin dans ce graphe entre une solution vierge (dans laquelle les taxis ne traitent pas de clients) et une solution optimale.

7.5.1 Sans batterie ni partage ni congestion

Dans un premier temps, ignorons les contraintes de charge, supposons qu'aucun client n'accepte de partage de course, et supposons que la congestion est constante. Montrons qu'il existe un chemin de la solution vierge s, dans laquelle les taxis sont stationnés de t_0 à t_{fin}, vers une solution optimale S.

A partir de s, on insère avec R^+ pour chaque taxi les requêtes qu'il traite dans S, en commençant par celles avec l'instant de départ le plus faible dans S. Lorsqu'une requête q est insérée durant le recuit, elle l'est le plus tôt possible, donc q est traitée au moins aussi tôt dans s que dans S. En conséquence, la course qui traite la requête suivante commence nécessairement au moins aussi tôt dans s que dans S. Puisque toutes les requêtes voient leurs contraintes de temps respectées, alors s finit bien par traiter toutes les requêtes que S traite.

Il existe donc un chemin de s vers S. Par ailleurs, ayant une solution s' quelconque, on peut obtenir s par retraits successifs de requêtes à l'aide de R^-. Il existe donc un chemin de n'importe quelle solution viable vers S.

A titre de remarque, l'algorithme glouton peut également trouver S en partant de s, à condition d'insérer les requêtes dans la solution dans le bon ordre (chronologiquement, par heure de départ dans S) et de choisir le bon taxi pour chaque requête (pas de règle précise).

7.5.2 Sans batterie ni congestion

Considérons maintenant le cas où certains clients acceptent le partage de course. Montrons que le recuit n'est pas toujours capable de trouver une solution optimale dans ce cas, même avec un seul taxi.

Prenons le cas particulier d'un edtt dans lequel 4 clients sont traités, qu'on appelle A, B, C et D. Les clients A, B et C sont traités avant un instant t, qui est l'instant auquel le début du traitement de D doit impérativement commencer. Montrons qu'un tel edtt peut être impossible à obtenir avec le recuit à partir d'une solution vierge à l'aide du contre-exemple ayant les caractéristiques suivantes. Les origines et destinations de ces requêtes sont placées dans un plan euclidien. La durée d'un trajet est égale à sa longueur (distance), sauf pour D, qui est à une distance égale de tout point. La figure 7.2 présente la disposition de ces origines et destinations ainsi que la course passant par tous ces points en minimisant la durée de parcours.

Nous avons testé exhaustivement tous les ordres d'insertions des requêtes A, B et C dans un edtt avec R^+, et nous avons ainsi obtenu les trois edtt de la figure 7.3. Il n'existe pas d'autre edtt qui puisse être obtenu avec R^+ (L'ordre d'insertion des deux premières requêtes peut être interverti sans conséquence). Dans tous les edtt générés avec R^+, la distance parcourue est supérieure à celle parcourue dans la figure 7.2. Si la requête D doit commencer

Requête	Origine		Destination	
	x	y	x	y
A	21	2	61	26
B	26	0	0	16
C	62	51	16	20

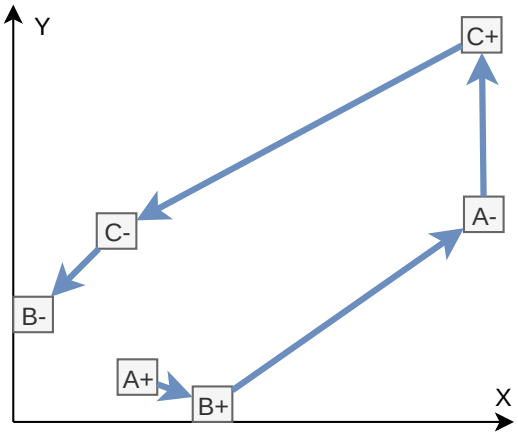


Figure 7.2 – *A gauche, un tableau indiquant les coordonnées spatiales de l'origine et la destination de 3 requêtes. A droite, les coordonnées spatiales d'un taxi durant une course partagée de longueur minimale traitant ces requêtes. La course commence au stop A+ (origine de A) et finit à B- (destination de B).*

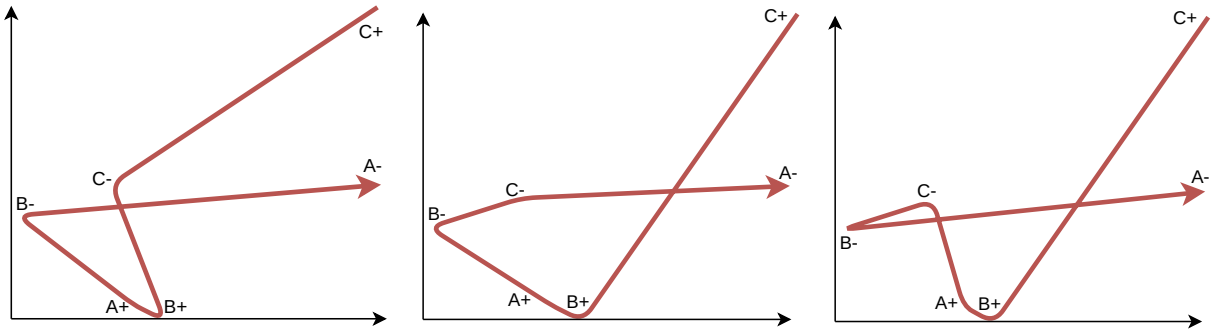


Figure 7.3 – *Courses obtenues en insérant les requêtes A, B, et C avec R^+ dans un edtt, en fonction de l'ordre d'insertion. De gauche à droite, l'ordre d'insertion des requêtes est ABC, ACB et BCA.*

immédiatement après la course de la figure 7.2, alors on conclut qu'il n'est pas possible d'obtenir un edtt traitant les quatre requêtes si elles sont insérées avec R^+ dans un edtt sans requête, quel que soit l'ordre d'insertion.

En utilisant à la fois R^+ et R^- pour insérer et retirer des requêtes, il n'est pas non plus possible de générer la course de la figure 7.2. Un parcours en largeur dans le graphe des voisinages à partir d'une solution vierge et avec ces deux mouvements a montré que seuls les trois edtt de la figure 7.3 traitent trois requêtes parmi les solutions qui peuvent être atteintes depuis la solution vierge.

Les mouvements complexes consistant en une suite de mouvements élémentaires (R^+ et R^-), on peut donc conclure que le graphe des voisinages n'est pas toujours connexe, et notamment qu'une solution optimale ne peut pas toujours être obtenue à partir d'une solution vierge, même lorsque la congestion est constante et que des charges ne sont jamais nécessaires.

Il s'agit donc d'un point améliorable de notre recuit. Pour rendre le graphe des voisinages connexe, on pourrait simplement envisager à chaque insertion de requête dans un edtt tous les ordres possibles pour les requêtes dans l'edtt. Cela rendrait le graphe des voisinages connexe lorsque la congestion est constante et les charges inutiles. Il faudrait donc sans

doute des modifications supplémentaires de nos mouvements pour rendre le graphe des voisinages connexe.

Pour un recuit durant des mois, des mouvements ainsi enrichis présenteraient sans doute de meilleurs résultats. Si le temps de calcul permis pour le recuit est plus limité, il est cependant possible qu'ils diminuent au contraire la qualité de la solution, en particulier compte-tenu de la taille des instances que l'on souhaite traiter, comme nous l'avons mentionné dans le chapitre 5.

7.6 Conclusion

Nous avons donc implémenté un recuit simulé ayant les caractéristiques suivantes. La température évolue après chaque mouvement. Les mouvements consistent en des insertions et des retraits de requêtes, les insertions et de retrait de charges peuvent avoir lieu, mais il n'existe pas de mouvement spécifiquement conçu pour les modifier. Les mouvements ayant une complexité en temps variable, la variation de la température est dépendante non pas du nombre de mouvements effectués mais plutôt du temps de calcul écoulé.

Ayant présenté les méthodes qui ont été implémentées, présentons maintenant les instances sur lesquelles elles ont été testées.

Chapitre 8

Création d'instances de simulation

Un aspect qui a paru essentiel pour cette thèse est l'usage de données réelles. L'idée principale qui a guidé la création des instances de test pour les heuristiques développées était que ces instances devraient être réalistes, représentatives des situations auxquelles seront confrontés des taxis autonomes électriques partagés, et, autant que faire se peut, utilisant des données réelles. Qui plus est, des instances de grandes tailles ont semblé préférables. D'une part, elles permettent d'évaluer la scalabilité des méthodes développées, et d'autre part, la gestion de la recharge électrique d'une flotte doit être observée sur une longue période. En effet, s'il faut une journée entière pour vider la batterie d'un taxi, une simulation sur quelques heures aurait peu de sens pour juger de l'efficacité d'un système de gestion de la recharge. Et si l'une des spécificités du modèle étudié est que plusieurs taxis ne peuvent pas utiliser simultanément un même chargeur, avoir un faible nombre de taxis serait peu pertinent.

Enfin, puisque le nombre de taxis doit être suffisamment élevé, il suit que le nombre de requêtes par heure doit aussi être élevé. Une instance dans laquelle les taxis traitent moins de 10% des requêtes dans la solution optimale, ou une instance dans laquelle toutes les requêtes sont traitées alors que la majorité des taxis ne sont pas utilisées ne semble pas pertinente. Manipuler une telle instance peut être intéressant pour observer le comportement des méthodes implémentées dans des cas particuliers, mais ne devrait pas constituer le cas de base de nos études.

8.1 Ville

La première étape consiste à générer le graphe de la ville. Pour rappel, ce graphe est celui présenté dans la figure 3.1, qu'on présente à nouveau dans la figure 8.1.

Le processus que l'on va décrire ici peut être divisé en trois grandes étapes : récupérer une carte, la formater, et générer le graphe de la ville lui-même.

La boîte limite d'un ensemble de figures géométriques est le plus petit rectangle dont un côté est parallèle à l'axe des abscisses contenant intégralement toutes les figures (voir figure 8.2). Cette définition de boîte limite peut être étendue aux graphes dont les sommets ont des coordonnées spatiales (voir figure 8.2). Le sous-graphe induit par une boîte limite est celui qui contient un sommet donné si et seulement si les coordonnées de ce sommet sont dans la boîte limite.

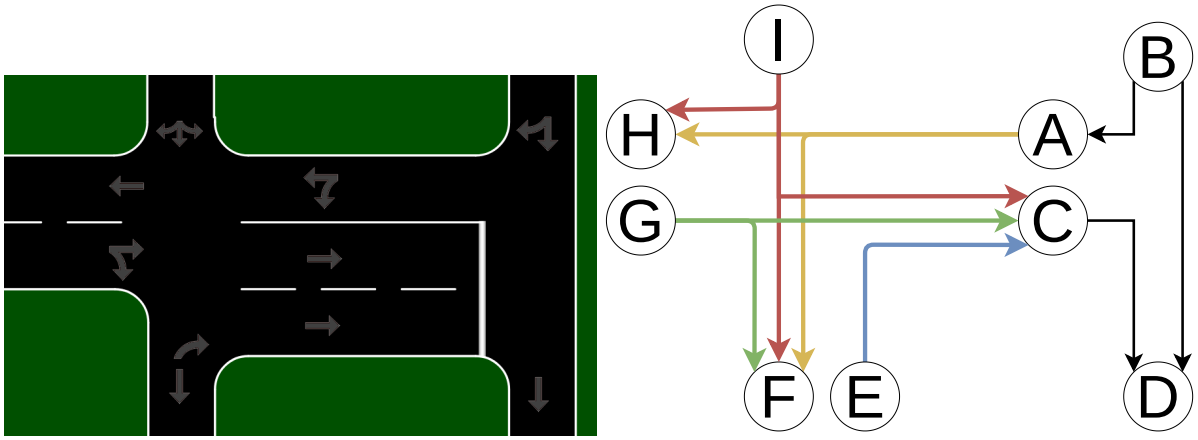


Figure 8.1 – *A gauche, représentation des rues d'une partie d'une ville. A droite, le graphe correspondant.*

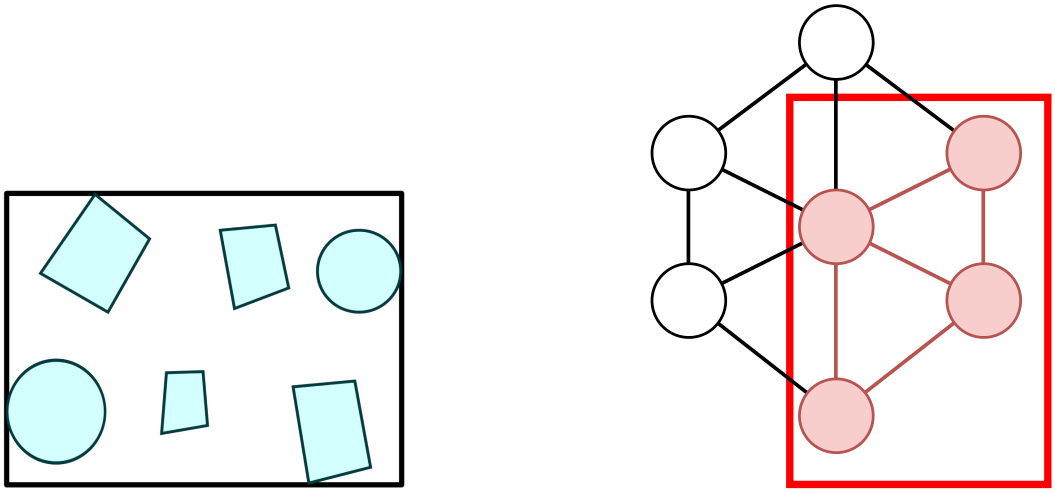


Figure 8.2 – *A gauche, un ensemble de figures (en cyan) et la boîte limite de cet ensemble. A droite, le rectangle rouge est une boîte limite. Les sommets et arêtes rouges forment le graphe induit par cette boîte limite.*

Lorsque les coordonnées sont exprimées avec le système de positionnement global (GPS), une boîte limite est un rectangle dans une projection orthogonale de la zone considérée.

Le point de départ pour générer le graphe de la ville est le site d'Open Street Map [website, 10] (OSM). Ce site présente une carte mondiale qui contient, entre autres informations, des informations sur les voies de circulations (piétonne, cyclable, automobile, ...). Un outil du site permet d'exporter une partie de cette carte contenue dans une boîte limite au format *.osm*.

La commande *netconvert* du logiciel SUMO [website, 11] (Simulator for Urban MObility) est ensuite utilisée pour formater ce fichier au format *net.xml*, et ne conserver que les informations relatives aux rues et intersections utilisables par des taxis. A ce stade, un réseau représentant ces rues peut être généré sur l'interface graphique de SUMO (voir figure 8.3).

SUMO est un simulateur microscopique, plutôt que macroscopique. Il est donc peu adapté à nos besoins, principalement dû au temps de calcul nécessaire pour trouver les plus courts chemins (Plus de détail sur ce point dans la suite de ce chapitre). Le même constat peut être fait de VISSIM [website, 12]. Il n'existe pas, à notre connaissance, de simulateur commercial

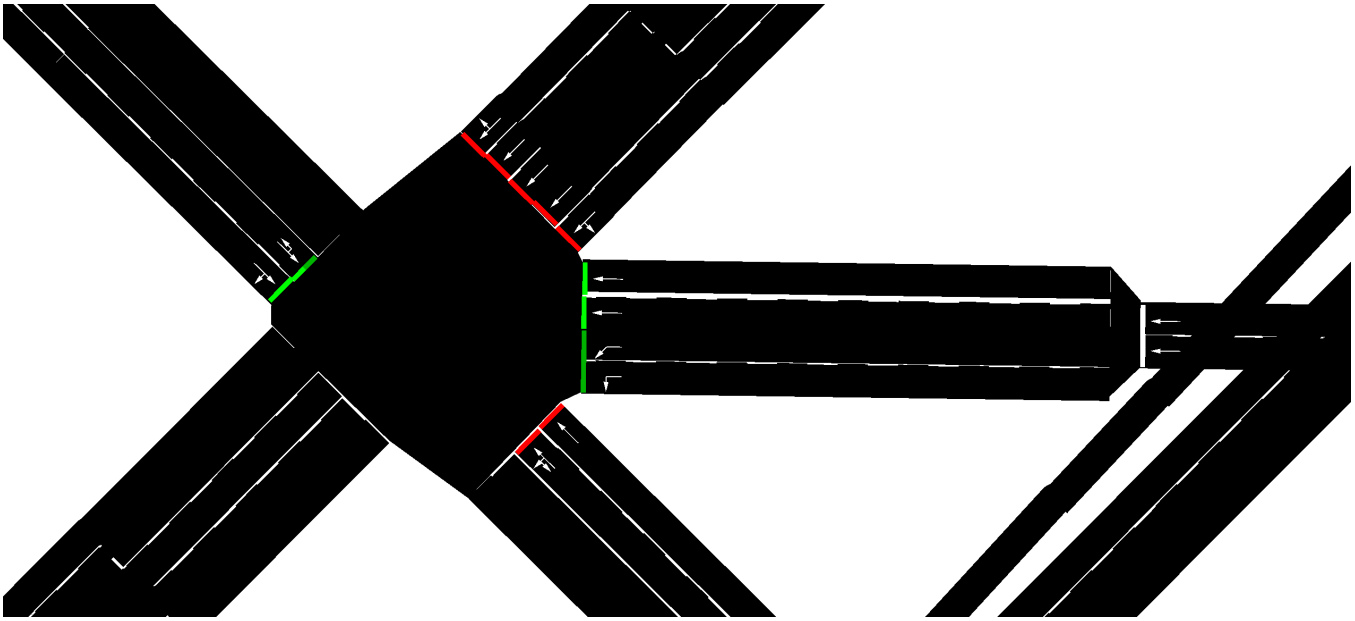


Figure 8.3 – *Une portion d'un réseau, visualisé dans SUMO.*

ou libre de mobilité urbaine à une échelle macroscopique. Il a donc fallu implémenter un tel simulateur.

Pour que le réseau .net.xml soit utilisable en dehors de SUMO, il faut le convertir au format .plain.xml, plus simple à manipuler. Un fichier contient une liste d'arcs, un autre contient une liste des sommets, ce qui permet de former un graphe.

Il est difficile de définir précisément ce que représente un arc dans ces fichiers. La définition la plus précise mais pas la plus claire est peut être la suivante. "Un arc représente une portion de la chaussée avec une forme, un sens, des coordonnées, utilisable par un ensemble de types de véhicules, et ayant des propriétés d'utilisation".

Le graphe généré à partir de ces arcs et sommets à donc, à ce stade, les caractéristiques suivantes :

1. Le graphe peut être faiblement connexe, voire non-connexe.
2. Il peut s'agir d'un multigraphe. Par exemple, deux arcs ayant un type différent dans SUMO peuvent avoir la même origine et destination.
3. Certains sommets ont un degré entrant ou sortant nul, généralement en bordure de la boîte limite utilisant dans OSM.
4. Il n'est pas planaire, même en l'absence de ponts et tunnels (e.g. celui de la figure 8.1).
5. Certains arcs ont une longueur de 1 mètre tandis que d'autres ont une longueur de 10 km.
6. Certains sommets ont exactement un arc entrant et un arc sortant.

Ayant les informations des arcs ci-dessus, il est possible d'obtenir le graphe de la ville correspondant. L'ensemble des sommets et des arcs est déjà connu, mais il faut encore établir, entre autres choses, la longueur et la vitesse de chaque arc. La vitesse moyenne des véhicules pour chaque arc est une des informations stockées dans les fichiers plain.xml. En ce qui concerne la longueur de chaque arc, elle peut être établie à l'aide de la forme des

arcs indiquée dans les fichiers plain.xml. La forme d'un arc est définie comme une suite de coordonnées GPS, qui indique l'ensemble des points par lesquels l'arc passe. La longueur d'un arc est donc la somme des distances euclidiennes entre chaque point successif.

On a ainsi instancié un graphe de la ville, et notamment attribué à chaque arc une vitesse et une longueur. Pour rendre ce graphe valide dans le cadre de notre modèle, on retire tous les arcs et sommets qui ne font pas partie de la composante fortement connexe contenant le plus de sommets. Lorsque plusieurs arcs partagent la même origine et même destination, on les supprime pour n'en garder qu'un, choisi au hasard, pour obtenir un graphe simple plutôt qu'un multigraphe.

On peut maintenant passer à l'instanciation des requêtes, taxis et chargeurs.

8.2 Requêtes

Les requêtes sont générées à partir d'historiques de trajets de taxis, obtenus sur internet :

New-York	http ://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
Porto	https ://www.kaggle.com/crailtap/taxi-trajectory
San-Francisco	https ://www.amodeus.science

Dans le cas de New-York, l'historique utilisé est celui des taxis jaunes de 2012, qui est le plus précis (dû à une plus faible réglementation en matière de vie privée avant 2013) et l'un des plus cohérents, *i.e.* contenant sensiblement moins d'anomalies évidentes que les jeux de données des années précédentes.

8.2.1 Rattachement au graphe de la ville

Dans un premier temps, ces jeux de données sont standardisés. Les données conservées sont les coordonnées GPS de l'origine et de la destination de chaque course, l'heure de départ, l'heure d'arrivée (utilisée pour calculer la congestion), et le nombre de passagers.

Il faut ensuite rattacher chaque origine et destination d'une course à un arc du graphe de SUMO. SUMO étant un logiciel libre (open source), nous avons créé une version alternative qui réutilise les fonctions de SUMO pour cette tâche. Une fonction en particulier permet de calculer la distance entre un arc et les coordonnées d'un point de l'espace. Cette méthode a été utilisée pour trouver, pour chaque origine et chaque destination l'arc le plus proche.

Ayant des centaines de milliers de courses (voire des millions) et des dizaines de milliers d'arcs, cette opération a été accélérée pour ne pas avoir à calculer la distance entre chaque paire origine/arc.

D'abord, la boîte limite de la ville est découpée en une matrice M de 1000x1000 boîtes limites de mêmes dimensions (approximativement des carrés de 50 mètres de côté). Une boîte limite est ici définie comme un ensemble de points. Pour chaque arc a, on génère sa boîte limite b, en bleu dans la figure 8.4. On détermine ensuite quelles boîtes de M sont superposées à b (ont une intersection non-vide avec b), ce qui se fait en temps constant (en jaune et orange dans la figure 8.4). Pour toute boîte $m \in M$ superposée à b, on regarde ensuite si elle contient l'arc a. En procédant ainsi, on établit rapidement pour chaque boîte de M quels arcs la traversent.

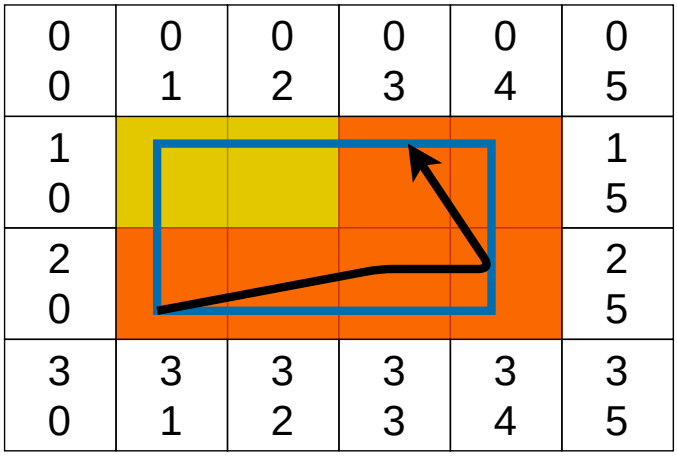


Figure 8.4 – *Division de la boîte limite d'une ville en 3×5 boîtes de même dimensions. La boîte limite b de l'arc a est en bleu. Les boîtes de M superposées à a sont en jaune ou orange. Les boîtes de M traversées par a sont en orange.*

Ville	longitude minimale	longitude maximale	latitude minimale	latitude maximale	Nombre de courses hors limites
New-York	-74.1	-73.75	40.57	40.87	3857 sur 420 000
Porto	-8.8	-8.2	40.1	40.4	2088 sur 1.7 millions
San-Francisco	-122.55	-122	37.2	38	3933 sur 463 000

Table 8.1 – *Boîtes limites des 3 villes étudiées.*

Ayant un point p avec des coordonnées GPS données, pour trouver l'arc le plus proche, la méthode suivante est utilisée. On liste toutes les boîtes de M susceptibles de contenir un arc à une distance inférieure à 100 mètres de p. S'il n'existe pas d'arcs à moins de 100 mètres de p, la course associée à p est considérée comme inutilisable. Sinon, on associe p à l'arc le plus proche.

Avec cette méthode, on peut ainsi associer à chaque course une origine et une destination dans le graphe de la ville.

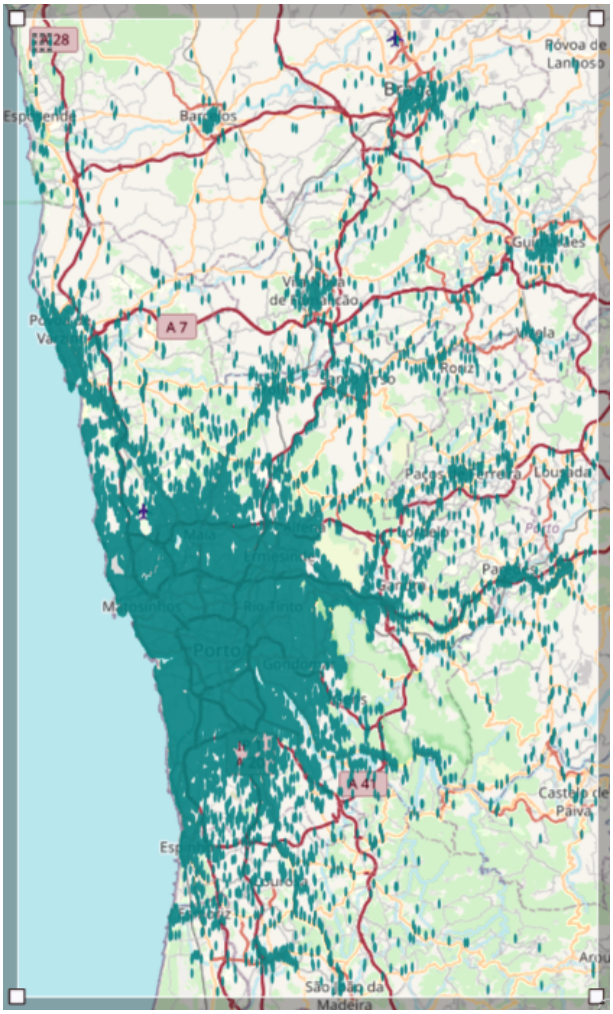
Pour chaque ville, la boîte limite a été choisie à l'aide des historiques de trajets de taxis. Ces boîtes limites sont de telle sorte que plus de 99.9% des requêtes générées à partir des historiques ont une origine (coordonnées GPS) et une destination dans cette boîte limite, avec l'aire la plus faible possible. Voir table 8.1.

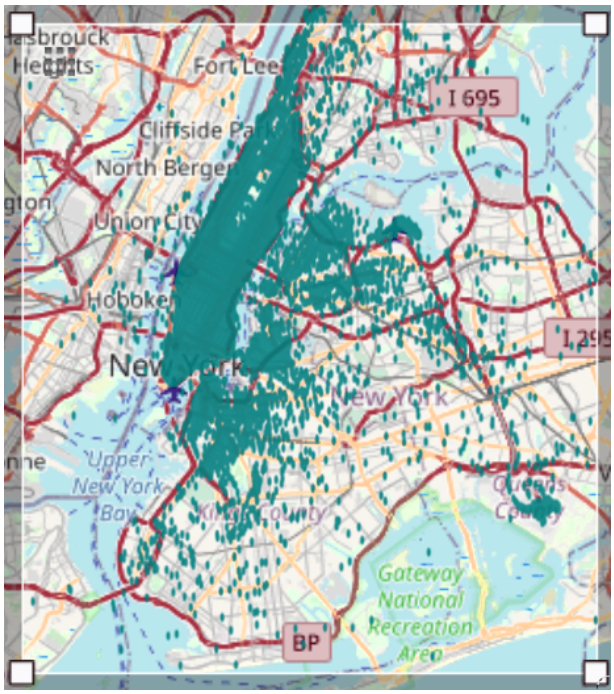
La figure 8.5 présente ces zones et la répartition géographique des origines et destination de courses.

Une course *standardisée* est caractérisée par une heure de départ et d'arrivée, une origine et une destination qui sont des sommets dans le graphe de la ville, et un nombre de passagers.

8.2.2 Génération de requête

Soit c une course standardisée. Pour générer une requête q à partir de c, on commence par lui attribuer la même origine et destination que c dans le graphe de la ville. L'heure idéale de départ de q est identique à l'heure observée de départ de c.





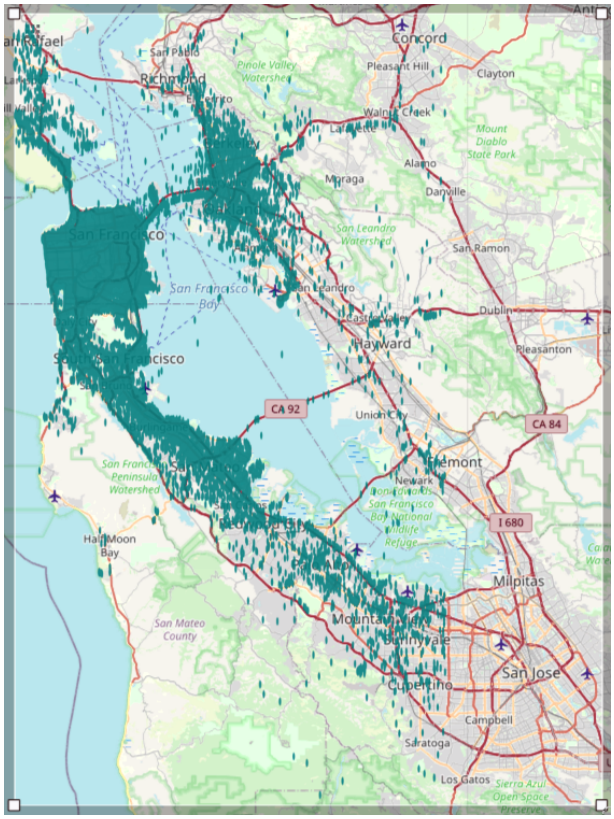


Figure 8.5 – *Répartition des origines et destinations des courses à Porto (gauche) sur 55x75km, New-York (haut droite) sur 35x35km et San Francisco (bas droite) sur 40x40km. Chaque point bleu clair indique soit une origine de course, soit une destination.*

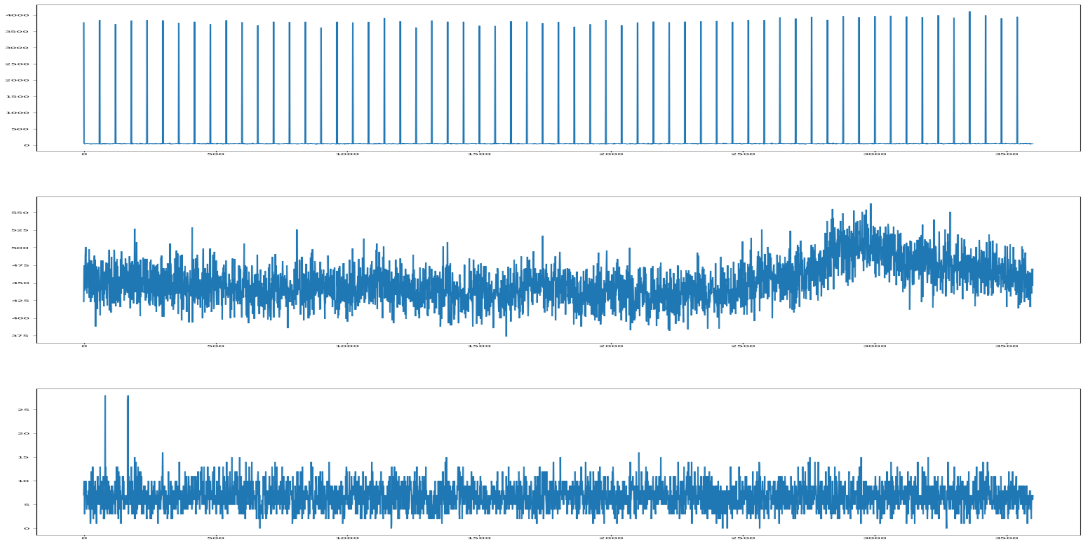


Figure 8.6 – *Distribution des courses dans le temps, à New-York, Porto, et San Francisco, dans cet ordre. La i-ème valeur de chaque courbe indique le nombre de courses commençant à la seconde i modulo 3600 secondes (une heure). A New-York, 99.0% de courses commencent à une minute ronde (e.g. 14h37m00s). Cela est vraisemblablement dû à la manière dont les données sont entrées dans la base de données.*

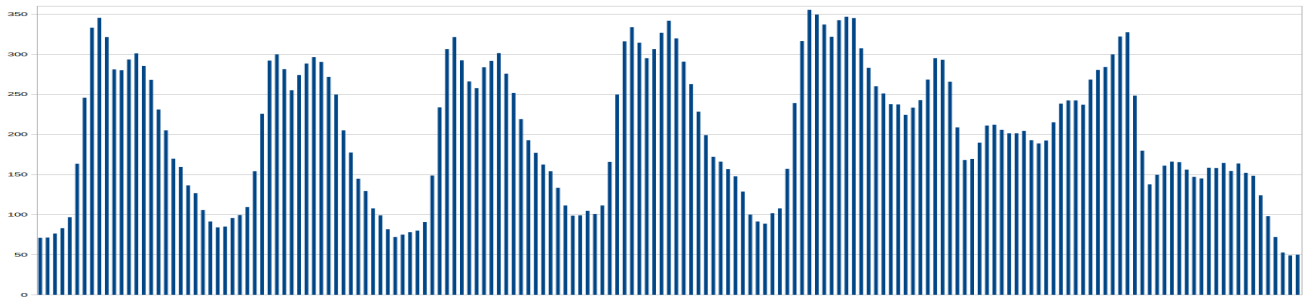


Figure 8.7 – *Requêtes traitées chaque heure, à Porto, durant une semaine, en commençant un lundi à 00 :00*

Du point de vue de l'heure de départ, les jeux de données semblent propres. La distribution sur le temps long est plausible (*e.g.* figure 8.7), et une analyse du temps court ne révèle pas d'anomalie majeure, comme le montre la figure 8.6. Un exemple d'anomalie possible est un nombre très important de requêtes formulées à minuit pile (à la seconde près), cette anomalie est courante dans les jeux de données ouverts sur internet.

A Porto et San Francisco, le nombre de passagers pour les courses n'est pas indiqué dans les historiques de trajets, un nombre aléatoire de passagers entre 1 et 4 est donc choisi suivant une loi uniforme pour chaque requête. Cette règle est aussi utilisée à New-York pour les courses avec 0 passager (erreur) ou les courses avec plus de passagers que la capacité du plus grand taxi.

La tolérance au retard est fixée par une constante (l'ensemble des constantes sont indiquées dans la section 8.4).

Dans nos instances, les requêtes sont divisées en quatre groupes : formulée avant le début de la simulation, formulée longtemps en avance, formulée peu en avance, au vol. La

proportion de requêtes dans chaque groupe dans le cas général a été determinée à l'aide de [Darbéra, 2010].

[Darbéra, 2010] a étudié dans la réalité la proportion des usagers obtenant un taxi par une réservation la veille, un hélage dans la rue, ou un appel téléphonique quelques minutes plus tôt. Cette proportion varie largement d'une ville a une autre : 90% de hélage à New-York contre 25% à Stockholm. Ces chiffres datent de 2010, avant l'apparition des smartphones et des compagnies de *ride-sharing*. Le pourcentage de hélage a probablement diminué depuis. La proportion de requêtes formulées avec quelques dizaines de minutes d'avance est surestimé dans nos instances pour simuler une augmentation du nombre de smartphones.

La proportion de chacun des 4 groupes présentés (au vol, réservation longtemps en avance, ...) dans nos simulations est indiqué dans la section 8.4, de même que la différence entre "longtemps en avance" et "peu en avance".

Dans nos simulations, nous ne prenons pas les jeux de données entiers. Après avoir généré toutes les requêtes possibles à partir des historiques de trajet de taxi, on en sélectionne un certain nombre suivant une loi uniforme.

Les caractéristiques des requêtes sont utilisées pour déterminer la congestion, mais aussi pour placer les taxis et chargeurs.

8.3 Autres

8.3.1 Plus courts chemins et congestion

Comme il a été mentionné dans la section 3.1.1, l'ensemble des plus courts chemins peut être précalculé. Dans nos instances, la taille de la matrice correspondante est de l'ordre d'une quinzaine de giga-octets. Dans cette matrice, les lignes et colonnes associées à des rues ne contenant ni l'origine ni la destination d'une requête ne nous intéresse pas, donc nous les oublions. Pour cette raison, les taxis et chargeurs ne peuvent être placés que dans des rues contenant l'origine ou la destination d'une requête. Cette réduction de la matrice fait passer sa taille à $2 \sim 3$ giga-octets dans les instances considérées.

A San Francisco et New-York, la congestion est calculée à partir des durées des trajets de taxis de leur historique respectif. Soit q une requête généré à partir d'un trajet r :

— On note d_1 la durée de r.

— On calcule d_2 la durée du plus court chemin de l'origine à la destination de q, estimée par SUMO.

— On note $c_r = \frac{d_2}{d_1}$ le rapport de congestion de r.

— Entre t_0 et t_{fin}, le temps est découpé en intervalles contigus de 15 minutes. On associe à chaque intervalle l'ensemble des trajets de taxis commençant durant ce bloc.

— La valeur de congestion de la ville dans un intervalle donné est le rapport de congestion moyen des trajets qui lui sont associés.

A Porto, la durée des trajets de taxis est inconnue, donc on suppose la congestion constante.

8.3.2 Taxis et chargeurs

La consommation des taxis par mètre augmente de 1 % par kilomètre/heure au dessus de 10 (vitesse de croisière).

Les taxis ne peuvent pas se recharger avant l'heure idéale de départ de la première requête ou une heure après l'heure idéale de départ de la dernière requête. Les valeurs exactes des tailles de batteries et leur consommation sont basées sur celles des voitures électriques actuelles (voir section 8.4).

Nos simulations durant plusieurs jours, l'emplacement initial des taxis à peu d'influence sur la valeur de la solution trouvée. Le placement des chargeurs a été l'objet de plus d'interrogations. Il a été envisagé de placer les chargeurs en suivant les préconisations de [Lam et al., 2014, Cui et al., 2019], pour couvrir les villes d'une manière optimale (sous certains aspects), et garantir que les chargeurs soient en bon nombre là où la population en a besoin.

En pratique, cette approche a ses limites. Non seulement il faut connaître les besoins de la population, avoir ces données, mais cette approche ignore en plus les multiples contraintes qui limitent la modification de la voirie dans les villes. Au-delà des contraintes classiques (monument historique classé, opinion des riverains), certaines contraintes sont propres aux chargeurs. La création de stations de recharge est ainsi limitée dans certains pays, et requiert l'autorisation des pompiers locaux. Une hypothèse paraissant vraisemblable est que les taxis et chargeurs devraient plutôt être proches des lieux dans lesquels des clients potentiels sont, mais que leur placement précis devrait être plutôt aléatoire.

Seules les rues contenant au moins une origine ou destination de requête peuvent être l'origine d'un taxi, ou contenir un chargeur.

Parmi ces rues, la moitié des taxis et chargeurs sont placés sur des rues aléatoires, en suivant une distribution uniforme. L'autre moitié est placée de la manière suivante : pour chaque rue, on compte le nombre de requêtes finissant ou commençant dans cette rue : la valeur obtenue est la valeur de la rue. On place ensuite un taxi dans chaque rue, en commençant par les rues de plus forte valeur, jusqu'à ce qu'il n'y ait plus de taxi ou chargeur à placer.

8.4 Constantes

Dans le chapitre suivant, voici les paramètres qui ont été utilisées, sauf mention contraire.
Constantes générales :

- Constantes de temps
  - Durée des arrêts lors des courses = 30 s
  - Durée de l'arrimage à un chargeur = 2 min
  - Durée de la recherche d'une place de stationnement = 15 min
  - Retard maximal toléré = 10 min + 20% de la durée minimale de la course traitant la requête (cette durée est souvent augmentée pour le problème dynamique)
  - Durée entre l'heure de réservation et le départ idéal (requête formulée longtemps en avance) : [1h, 10h]
  - Durée entre l'heure de réservation et le départ idéal (requête formulée peu en avance) : [10 min, 30 min]

    - Pour réduire le nombre de requêtes impossible à traiter, les taxis peuvent commencer à bouger deux heures avant l'heure de départ idéal de la première requête (chronologiquement), et doivent être stationnés deux heures après l'heure de départ idéal de la dernière requête (chronologiquement). Les taxis ne peuvent pas se recharger durant ces buffers.
- Vitesse de croisière (utilisée pour calculer la consommation énergétique lors d'une phase de recherche de place de stationnement : 10km/h
- Constantes de batterie
    - Niveau de batterie initial des taxis : 40-100% (tirage aléatoire uniforme pour chaque taxi dans cet intervalle)
    - Niveau de batterie final minimal : $\geq$ 60%
    - Pourcentage de charge minimal : 1%
    - Pourcentage de charge interrompue minimal : 1% (dans le problème dynamique, on peut interdire un taxi d'interrompre une charge pour aller traiter un usager si celle-ci devient trop courte en conséquence.
- Constantes des requêtes
    - Pourcentage d'usagers acceptant de partager leur taxi : 100%
    - Les clients formulant leur requête avant le début de la simulation acceptent un retard au départ : VRAI
    - Requêtes formulées avant le début de la simulation : 10%
    - Requêtes "au vol" = 20%
    - Requêtes formulées longtemps en avance : 35%
    - Requêtes formulées peu en avance : 35%
- Décharge par mètre des taxis = 0.15 Wh (ce qui se traduit par 15kWh/100km)
- Décharge par seconde des taxis = 0.5 Wh (soit 1.8kWh/h)

8.5 Instances dures

Pour tester les limites des méthodes, des instances "dures" ont été générées. Ces instances ont été construites avec deux idées clefs. Dans ces instances, il existe une solution traitant toutes les requêtes, et les taxis n'ont pas de temps mort. L'objectif est de déterminer les performances des heuristiques développées dans des cas où il est possible mais très difficile de traiter toutes les requêtes.

Créer une de ces instances se fait de la manière suivante. On prend d'abord une instance normale, puis on en retire toutes les requêtes et tous les chargeurs.

On va ensuite générer l'ensemble des requêtes et chargeurs en même temps qu'une solution qui traite toutes les requêtes. Soit e un edtt de cette solution. e est initialement vide.

e est rempli en alternant courses et charges. Une première course est ajoutée, traitant une unique requête a. La suite des stops de la course est noté (a^+, a^-). Ensuite, tant que

la course ne vide pas à plus de 90% la batterie, on ajoute une requête dont la montée est avant le dernier stop, et la descente après. Donc avec l'addition d'une requête b, la suite des stops de la course devient (a^+, b^+, a^-, b^-). En lui ajoutant une requête c de plus, elle devient $(a^+, b^+, a^-, c^+, b^-, c^-)$.

Soit k un entier naturel positif aléatoire inférieur à la capacité du taxi. Soit n la capacité du taxi. Le nombre de passagers des requêtes impaires est k, celui des requêtes paires est $n - k$.

Lorsque la batterie du taxi est vide, une charge est ajoutée, commençant immédiatement après la course, sans trajet à vide entre les deux. Il s'agit d'une charge complète.

L'algorithme 3 présente le processus de construction d'instances dures.

algorithme 3 – Génération d'edtt durant la génération d'instances dures

ENTREE : Un taxi x
SORTIE : Un edtt e pour x

DEBUT :
Créer un edtt e, ne contenant qu'une phase de parking de t_0 et durant deux heures, à l'origine de x.
Dans la suite de cet algorithme, a désigne la dernière activité de e
t_{stop} désigne la fin de la simulation moins deux heures.
tant que a finit avant t_{stop} **faire**
  si a est une charge ou un stationnement **alors**
    Créer une requête q ne tolérant presque aucun retard.
    q a un nombre aléatoire de passagers.
    L'origine de q est la destination de a
    l'heure idéale de départ de q est la fin de a.
    Créer une course qui traite uniquement q, et l'insérer après a
  sinon
    tant que a finit avec plus de 10% de batterie et avant t_{stop} **faire**
      Soient s et s' les deux derniers arrêts de a, qui est nécessairement une course.
      Créer une requête q sans caractéristiques définies.
      Dans a, insérer la montée de q entre s et s', et la descente de q après s'
      Choisir une origine pour q sur le plus court chemin de s vers s'
      Choisir une destination pour q, avec idéalement s' sur le chemin.
      Soit k le nombre de passagers qui descendent à s'.
      Le nombre de passagers de q est égal à $places_x - k$
    Créer un chargeur là où x se trouve à la fin de e
    Ajouter une charge complète à la fin de e à ce chargeur
Ajouter une phase de stationnement de la fin de a jusqu'à la fin de la simulation
renvoie e

Ayant créé un edtt pour chaque taxi, la génération de l'instance ne demande plus qu'à extraire de ces edtt toutes les requêtes et tous les chargeurs. On sait qu'il existe une solution traitant toutes les requêtes, et on sait que dans cette solution, les taxis sont souvent très pleins, donc il existe vraisemblablement peu d'attributions de requêtes à des taxis qui permettent d'obtenir une solution optimale.

Chapitre 9

Résultats

Les méthodes présentées ont été codées en C++. Les expérimentations ont été menées sur un DELL Latitude 7490, qui contient notamment un processeur 4 cœurs.

Les paramètres des instances varient évidemment, mais elles possèdent un socle commun. Dans la suite de ce chapitre, voici les valeurs et caractéristiques des instances, sauf indication contraire :

— Un taxi a 5 places, consomme 15kWh/100km plus 1.8kW hors stationnement, et a une batterie de $72kWh$ de capacité. Il y a 30 taxis.

— Un chargeur a une puissance de 18kW.

— Dans l'algorithme glouton, le taxi le plus proche est envoyé lorsque plusieurs peuvent traiter une même requête. Les requêtes sont triées chronologiquement (heure idéale de départ dans le problème statique, heure de formulation dans le problème dynamique).

— Une solution dont la valeur indiquée est N signifie que la valeur totale des requêtes traitées par cette solution est $N\%$ de la valeur totale des requêtes en entrée, sachant par ailleurs que l'existence d'une solution admissible traitant toutes les requêtes n'est pas garantie.

— À New-York, les requêtes ont lieu en avril 2012, entre le dimanche premier et le lundi 9 (inclus). Aucune célébration n'a significativement perturbé la circulation cette semaine.

— À Porto, les requêtes ont lieu en juillet 2013, entre le mercredi 3 et mardi 9.

— À San Francisco, les requêtes ont lieu en Juin 2008, entre dimanche premier et lundi 9.

Dans la suite de ce chapitre, on distingue le temps d'une instance, qui est l'ensemble des instants entre t_0 et t_{fin} (voir section 4.1) et le temps d'exécution d'une instance, qui est la durée nécessaire pour exécuter un algorithme, comme le recuit simulé.

9.1 Validité opérationnelle pour le problème dynamique

La première étape des analyses a consisté à tester que l'algorithme glouton a des performances en temps qui le rendent viable pour une utilisation en temps réel. Ensuite, il a été vérifié que la solution qu'il trouve n'est jamais très mauvaise, ou en tout cas qu'il n'est pas facile d'établir que la solution trouvée est mauvaise.

Ville	Requêtes	temps (minutes)	Requêtes par seconde et par processeur (cœur)	Nombre de taxis	Nombre de chargeurs
New-York	30 000	24 (2)	5 (50)	60	12
San Francisco	30 000	14	9	60	12
Porto	30 000	27	5	60	12
New-York	100 000	198	2	200	40
New-York	10 000	3	800	20	4

Table 9.1 – *Temps de calcul requis par l'algorithme glouton dans le problème statique. Dans ces instances, le nombre de taxis est adapté pour qu'environ 95% des requêtes soient traitées. Les chiffres entre parenthèses concernent le problème dynamique, avec réservation 10 minutes en avance. Dans la solution de cette instance dynamique, 63% des requêtes sont traitées.*

9.1.1 Scalabilité

Comme il a été mentionné à plusieurs reprises dans ce mémoire, il était attendu qu'une méthode implémentée pour le problème dynamique soit capable de décider en temps réel de l'acceptation ou du rejet des requêtes. Le recuit simulé n'est pas utilisable dans le cas dynamique, dans la mesure où il est susceptible de retirer de la solution des requêtes qui ont déjà été acceptées. Montrons donc que l'algorithme glouton est scalable.

Comme il a été mentionné dans la section 6.4.4, l'algorithme glouton est parallélisable par taxi, *i.e.* il est possible d'allouer un processeur pour chaque taxi, et tester si une requête peut être insérée dans son edtt sans savoir comment les autres taxis envisagent de traiter la requête. Pour cette raison, pour déterminer si l'algorithme glouton est utilisable en temps réel, il suffit de montrer qu'il est capable de tester l'insertion d'une unique requête dans un unique edtt en quelques secondes.

Dans nos expérimentations, nous avons toujours constaté que l'algorithme glouton était capable de cela, même dans les cas les plus défavorables. Dans le pire cas de figure, les clients réservent longtemps en avance, donc une grande partie des edtt peut être modifiée (pour cette raison, le temps de calcul est sensiblement plus important dans le problème statique. De plus, le taux d'acceptation des requêtes est élevé (chaque requête acceptée augmente le nombre d'activités dans un edtt, ce qui augmente le temps de calcul pour les insertions de charges et des requêtes suivantes). La table 9.1 présente le temps de calcul pour plusieurs instances. Les requêtes sont insérées par l'algorithme glouton par ordre chronologique et le taxi le plus proche est choisi lorsque plusieurs sont capables de traiter une requête.

Dans toutes les villes, plusieurs requêtes sont insérées par seconde par processeur. Multiplier par 3 ce nombre le requête multiplie par 10 le temps de calcul, la complexité effective de l'algorithme glouton dans ces instances semble donc quadratique (tout en permettant une augmentation linéaire du nombre de processeurs utilisables). L'algorithme glouton est donc largement scalable, sans même compter le fait que les insertions sont sensiblement plus rapides dans le problème dynamique.

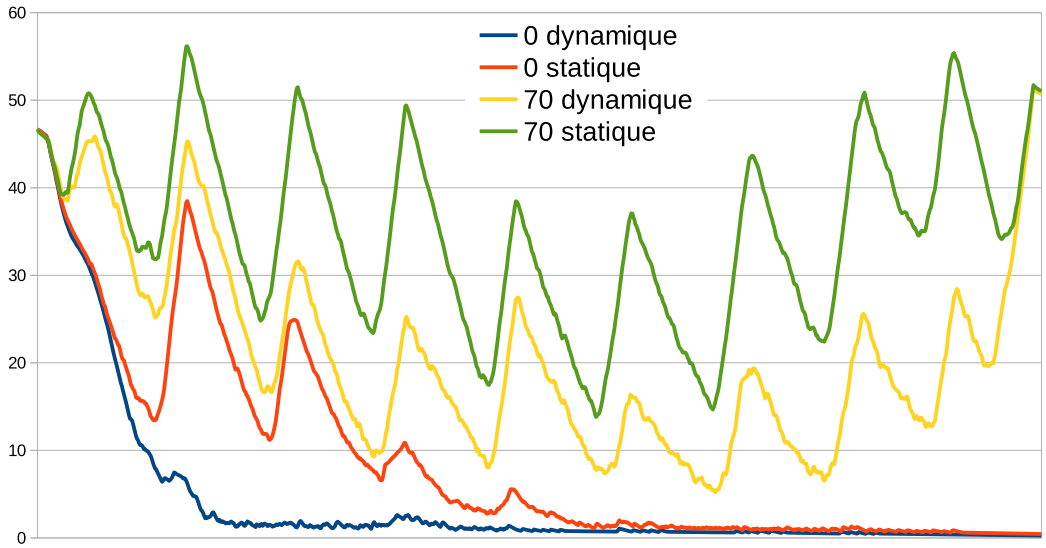


Figure 9.1 – *Énergie moyenne des taxis (en kWh) au cours du temps d'une instance (9 jours) à New-York. "70 statique" signifie que la courbe a été obtenue sur une instance du problème statique dans laquelle les taxis doivent finir avec au moins 70% de batterie. Dans les instances dynamiques, les usagers appellent entre 10 et 50 minutes avant leur heure idéale de départ. Dans les instances statiques, les requêtes sont insérées par valeur décroissante.*

9.1.2 Qualité crédible

L'algorithme glouton trouve toujours une solution admissible, à condition que l'insertion initiale de charge réussisse. Montrons dans cette section qu'une solution trouvée par l'algorithme glouton a une valeur dont on peut raisonnablement supposer qu'elles est dans le même ordre de grandeur que l'optimum.

Expérimentalement, nous avons établi que les taxis passent relativement peu de temps stationnés dans toutes nos instances (moins de 20% du temps). Ce n'était pas le cas dans notre première implémentation de l'algorithme glouton. Si le niveau minimal de batterie en fin d'edtt (w_{fin}) avait une valeur proche de 0, alors on observait la chose suivante :

— Les taxis ne se rechargeaient que lorsqu'ils arrivaient complètement à court de batterie (aucune insertion de charge n'avait lieu tant qu'aucune contrainte de batterie n'était violée)

— Un taxi qui avait besoin de batterie n'avait pas suffisamment d'énergie pour aller se recharger.

— Ce taxi s'arrêtai donc définitivement de bouger dès qu'il aurait du se recharger.

Cet effet était accentué lorsque les requêtes étaient insérées chronologiquement. La figure 9.1 présente les résultats obtenus par l'algorithme glouton en fonction du niveau minimal final de batterie.

Ce défaut a été résolu en forçant les taxis à finir avec au moins 5% de batterie. Cela a un faible impact négatif sur la qualité de la solution, puisqu'une contrainte est ajoutée, mais règle complètement ce problème. Sur les instances qu'on considère, la qualité des solutions est très peu influencée par le niveau de batterie minimale en fin de simulation. Un

changement de cette valeur change généralement la qualité de la solution trouvée de moins de 1%.

Dans la suite de ce chapitre, les taxis commencent avec 40 à 100% de batterie, et finissent avec au moins 60% de batterie, ce qui implique une variation de charge presque nulle entre le début et la fin de la simulation (le niveau de charge moyen au début et à la fin est environ 70%).

Concernant les autres paramètres des instances, nous n'en avons pas trouvé faisant que l'algorithme glouton trouve des solutions absurdes (mais admissibles), à condition que la stratégie choisie soit bonne. Présentons maintenant un comparatif des différentes stratégies existantes.

9.2 Comparaison des stratégies pour l'algorithme glouton

Dans le problème statique, deux paramètres majeurs de l'algorithme glouton doivent être établis : L'ordre d'insertion des requêtes et le critère de sélection de taxi.

9.2.1 Acronymes

Dans la suite de ce mémoire, les principaux paramètres de l'algorithme glouton sont indiqués par une séquence de trois lettres. Le symbole '-' signifie "n'importe quelle lettre". La première indique quelle caractéristique des requêtes utiliser pour les trier :

R (Random) Pas de tri	**V** Valeur	**S** (Start) instant idéal de départ	**C** (Call) instant de réservati

La deuxième lettre indique si les requêtes sont triées par ordre croissant ou décroissant : soit **H** (*Highest first*) soit **L** (*Lowest first*). La troisième lettre indique le taxi choisi lorsque plusieurs taxis peuvent traiter une requête.

C *Closest :* taxi le plus proche (minimisant la durée du trajet à vide précédant la course, ou la durée du détour vers le stop de montée du client et vers le stop de descente du client dans le cas d'une course partagée).

R *Random :* aléatoire

M taxi avec le plus de batterie à l'instant de début de la course (Most battery)

L taxi avec le moins de batterie à l'instant de début de la course (Least battery)

I *Idle :* taxi qui a été inactif (sans course) le plus longtemps avant le début de la course

Donc par exemple, **VHR** désigne l'algorithme glouton, dans lequel les requêtes de plus forte valeur sont insérées les premières, et un taxi aléatoire est choisi lorsque plusieurs peuvent traiter une requête.

Lorsque l'algorithme glouton est appliqué au problème dynamique, les deux premières lettres sont nécessairement **CL**.

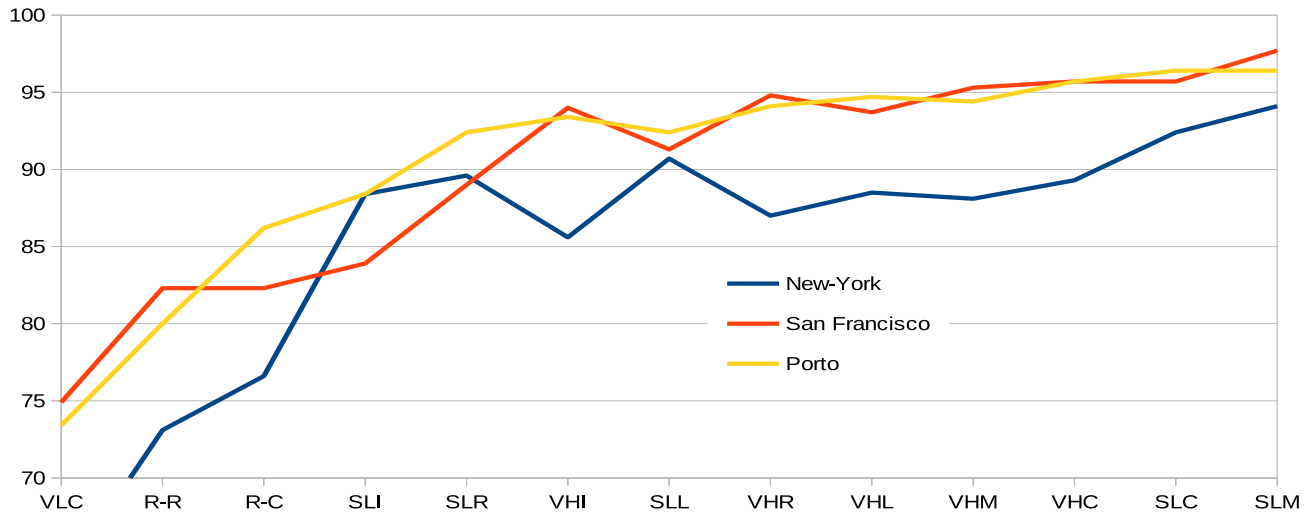


Figure 9.2 – *Valeur de la solution trouvée par l'algorithme glouton en fonction de la stratégie utilisée, dans des instances où l'infrastructure de recharge est limitée (voir section 9.2.2). Les jeu de paramètres sont triés par efficacité moyenne. Par exemple, SLM est à l'extrémité droite car c'est la stratégie qui a donné le meilleur résultat en moyenne dans les trois villes étudiées.*

9.2.2 Instances de test

Nous avons comparé ces stratégies dans le cas statique. Décrivons comment nous avons choisi le nombre de chargeurs et de requêtes dans les instances utilisées pour les comparer.

Nous avons d'abord cherché à établir comment évolue la valeur de la solution trouvée par l'algorithme glouton en fonction du nombre de chargeurs et de leur puissance. Cette relation est influencée par de nombreux paramètres (stratégie de l'algorithme glouton, taille des batteries des taxis, emplacement des chargeurs, ...). Nous avons observé qu'en règle générale, la valeur de la solution trouvée croît d'abord linéairement avec le nombre de chargeurs, puis un seuil est atteint à partir duquel ajouter de nouveaux chargeurs n'a qu'un effet marginal. Ce seuil est atteint à environ 1 chargeur pour 5 taxis.

Pour comparer les stratégies de l'algorithme glouton, nous avons distingué deux cas. Dans l'un, les taxis ont une batterie infinie. Il s'agit d'un cas extrême nous permettant d'évaluer la qualité des solutions lorsque l'infrastructure de recharge est surdimensionnée. Dans l'autre, le nombre de chargeurs est proche du seuil (1 chargeur pour 5 taxis). Dans le reste de ce chapitre, lorsque le nombre de chargeurs n'est pas mentionné, c'est que ce rapport est utilisé.

Concernant le nombre de requêtes, nous l'avons ajusté de sorte qu'avec la meilleure stratégie, la valeur de la solution est proche de 95 lorsque la meilleure stratégie est utilisée, avec des taxis à batterie finie.

9.2.3 Résultats

La figure 9.2 présente les performances de l'algorithme glouton dans les instances où le nombre de chargeurs est proche du seuil.

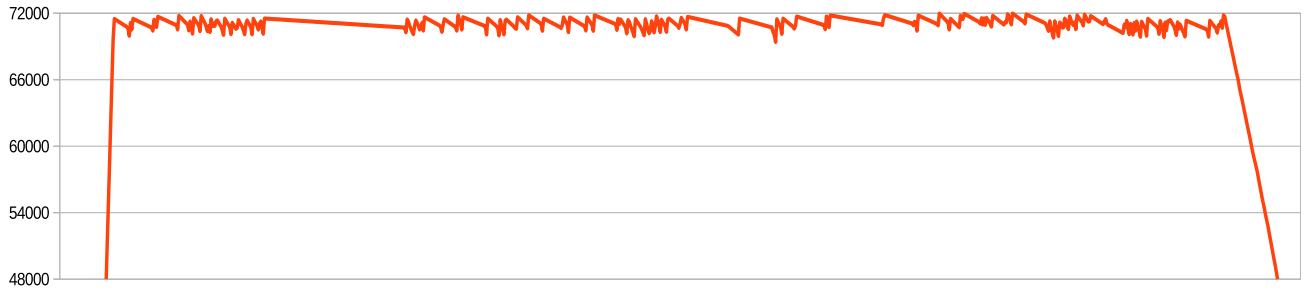


Figure 9.3 – *Niveau de batterie d'un taxi (en Wh) durant un edtt obtenu avec SHC. Tous les edtt obtenus avec SHC ne sont pas ainsi, mais cette forme est courante. Dans cet edtt, il y a donc une charge complète au début, une décharge à la fin (le niveau de batterie final du taxi est très légèrement supérieur au minimum requis), et aucune décharge de plus de 2% entre les deux. L'utilisation de l'infrastructure de recharge est très inefficace.*

Mauvaises stratégies

Les résultats de SH- (insertion par ordre chronologique inverse) ne sont pas visibles car ils sont largement inférieurs à ceux des autres méthodes ; la valeur de la solution trouvée est ~ 30 à New-York, ~ 50 à Porto, et ~ 70 à San Francisco. Cela est lié au niveau de batterie dans un edtt généré par SH-, dont la figure 9.3 montre un exemple. La figure 9.4 indique comment cet edtt a été bâti.

Insérer les requêtes de plus faible valeur en priorité (VL-) fournit une mauvaise solution, sans grande surprise.

La solution de mauvaise qualité de R-R (choix aléatoire de taxi, insertion des requêtes dans un ordre quelconque) n'est probablement pas causé par un tirage aléatoire défavorable. En effet, l'écart type est proche de 0.3 pour ces instances.

VH- et SL- (valeur et chronologique)

En moyenne, les différences de performances entre VH- (plus forte valeur d'abord) et SL- (ordre chronologique) sont assez mineures. On note toutefois que ces jeux de paramètres ont des performances fortement liées à l'instance dans laquelle elles sont testées (en particulier, pour VHI et SLL).

SLL VS SLM

SLL favorise un niveau hétérogène de batterie (plus un taxi à une batterie faible, plus on l'utilise et vide sa batterie), tandis que SLM favorise un niveau homogène. Dans les 3 villes, maintenir un niveau homogène semble préférable.

Cela reste vrai même dans une instance dans laquelle les requêtes sont distribuées uniformément dans le temps. Autrement dit, une stratégie qui fait que tous les taxis ont autant besoin de se recharger à tout instant est préférable à la fois dans des instances avec des accalmies (cycle jour-nuit) et dans des instances avec des arrivées constantes.

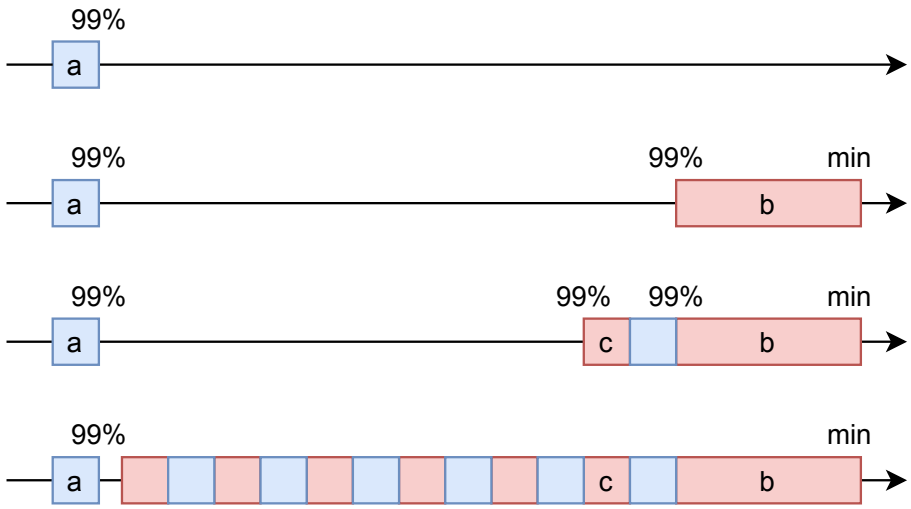


Figure 9.4 – *Formation d'un edtt avec SHC. Une charge complète (a) est initialement ajoutée, faisant monter à 99% le niveau de batterie (en incluant la consommation du stationnement qui suit cette charge). Des courses sont ensuite ajoutées par ordre chronologique inversé, et forment un bloc contigu (b). Le niveau de batterie à la fin de ce bloc est très légèrement supérieur au niveau minimal final. Si, après l'ajout d'une course c, le niveau de batterie au début de b est inférieur à 99%, alors l'edtt est invalide, car le niveau de batterie final devient trop bas. c n'est donc ajoutée avant b que si elle peut être suivie d'une charge qui donne autant d'énergie que c consomme. Par récurrence, on obtient un edtt qui alterne course et charge entre a et b.*

Meilleure méthode

Bien que SLM soit la stratégie dominante dans les instances de la figure 9.2, elle est fortement influencée par l'infrastructure de recharge et les caractéristiques énergétiques des taxis. Si les taxis ont une batterie infinie, SLM a un comportement similaire à SLR. La figure 9.5 présente les résultats obtenus sur les mêmes instances que dans la figure 9.2, mais avec des taxis ayant une batterie infinie.

Plutôt que SLM (SLR), c'est SLC qui se démarque dans ce cas de figure. SLC peut être moins efficace que VHC lorsque l'accès à l'infrastructure de recharge est très contraint. Prenons par exemple une instance de New-York avec 20 taxis et 6 chargeurs, et dans laquelle les chargeurs deviennent inutilisables pendant 12 heures toutes les 23 heures. Voici les valeurs des solutions obtenues par les différentes stratégies : SLC (58), SLM (61), et VHC (73).

Les figures 9.6 et 9.7 présentent les activités effectuées par les taxis dans la solution de VHC et de SLC, respectivement. Dans ces figures, on peut voir clairement qu'avec SLC, les taxis se chargent beaucoup moins durant le premier tiers de la solution (la portion de jaune est plus petite que dans la solution de VHC). SLC a un comportement court-termiste, au point qu'il y a des périodes de plusieurs heures durant lesquelles tous les taxis sont stationnés. Simultanément, il y a dans cette solution beaucoup plus de charges en fin de solution, et la proportion trajets à vide / course est plus élevée.

En conclusion, dans le PCE-ADARP statique, insérer les requêtes par ordre chronologique (SL) ou par valeur décroissante (VH) semble pour le mieux, et choisir l'une ou l'autre de ces méthodes devrait être fait empiriquement. Choisir quel taxi attribuer est aussi un paramètre pour lequel l'approche optimale dépend des caractéristiques précises de l'instance, les deux approches se démarquant du lot consistant à envoyer le taxi le plus proche (C) ou celui avec

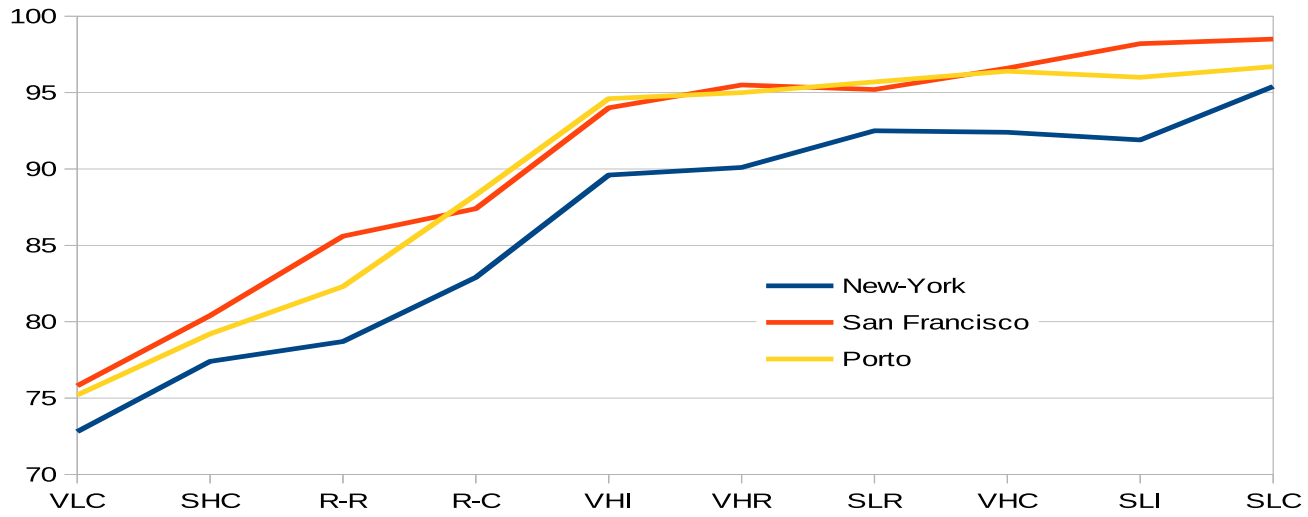


Figure 9.5 – *Valeur de la solution trouvée par l'algorithme glouton en fonction de l'ordre d'insertion des requêtes et de la stratégie de choix de taxi, dans des instances où les taxis ont une batterie infinie.*

le plus de batterie (M).

Cette conclusion amène à de nouvelles perspectives de recherche. Une technologie actuellement étudiée est celle du V2V *vehicle to vehicle*, qui consiste à permettre un transfert d'énergie d'un véhicule vers un autre. Puisque SLM favorise une homogénéisation du niveau des batteries, permettre à un taxi avec beaucoup d'énergie d'en donner pourrait améliorer la qualité des solutions trouvées lorsque l'accès aux chargeurs est limité.

Instances dures

Dans cette sous-section, les observations menées sur les instances de la section 8.5 sont présentées.

La manière dont ces instances sont construites favorise trop SLC, qui est presque garantie de trouver une solution traitant toutes les requêtes. Elles semblent en revanche pertinentes pour juger la qualité de VHC.

Lorsque la fenêtre de temps des clients est de 2 minutes, VHC obtient une solution d'une valeur comprise entre 60 et 65 point. On peut donc encore augmenter de moitié (x1.5) la qualité de la solution trouvée, sur des instances relativement réalistes.

9.3 Performances dans le problème dynamique

Comparons maintenant la valeur de la solution trouvée dans le problème dynamique à celle trouvée dans le problème statique.

Si les clients réservent avec une très grande avance (>24 heures), la solution trouvée par l'algorithme glouton dans le problème dynamique est presque la même que celle trouvée avec SL- dans le problème statique, sur la même instance.

Montrons maintenant comment évolue la qualité de la solution trouvée par l'algorithme glouton en fonction du degré d'avance avec lequel les clients réservent. Dans une instance

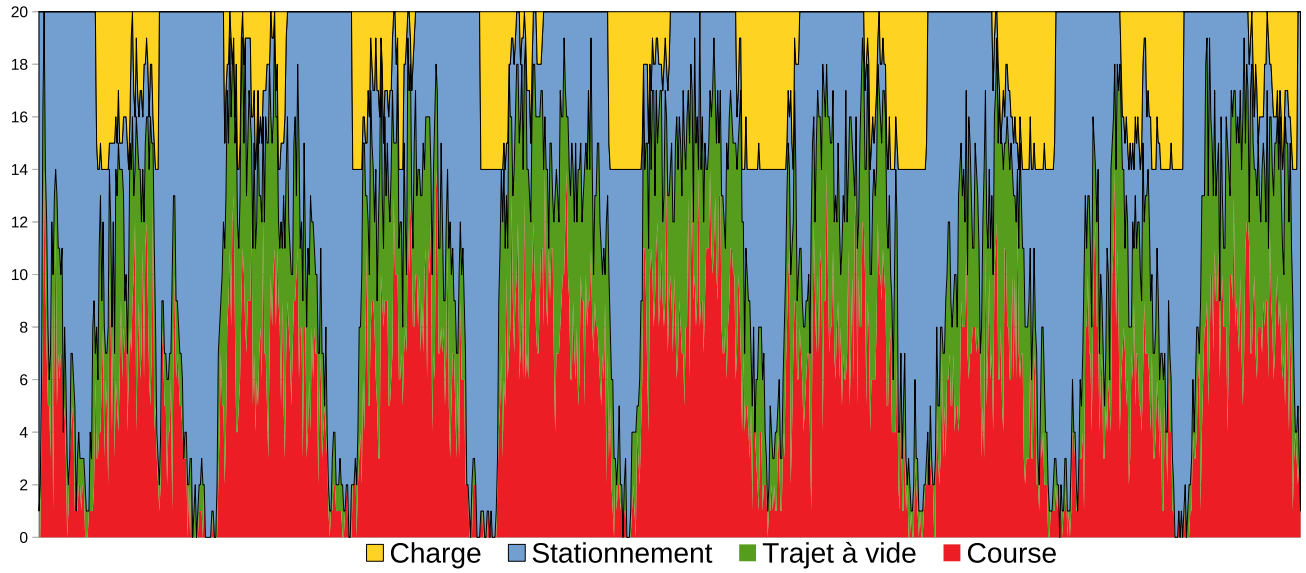


Figure 9.6 – *Chaque colonne indique le nombre de taxis en train d'effectuer chaque type d'activité à chaque quart d'heure de t_0 à t_{fin}. Les chargeurs deviennent indisponibles pendant 12 heures toutes les 23 heures. Les edtt sont générés par VHC.*

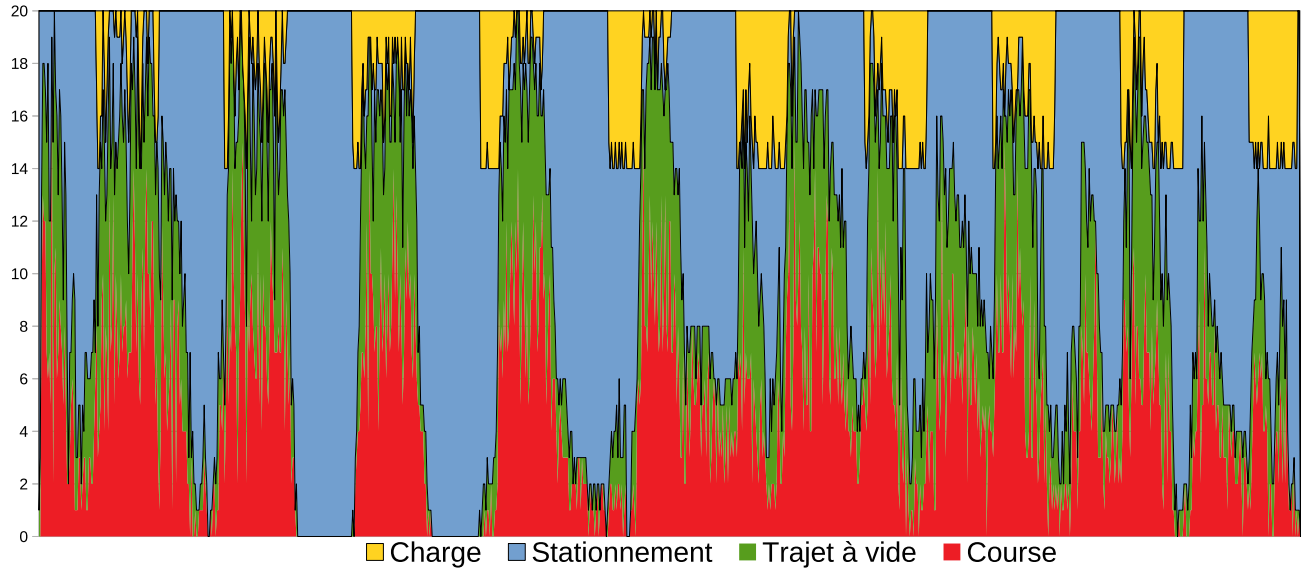


Figure 9.7 – *Ici, les edtt sont générés par SLC.*

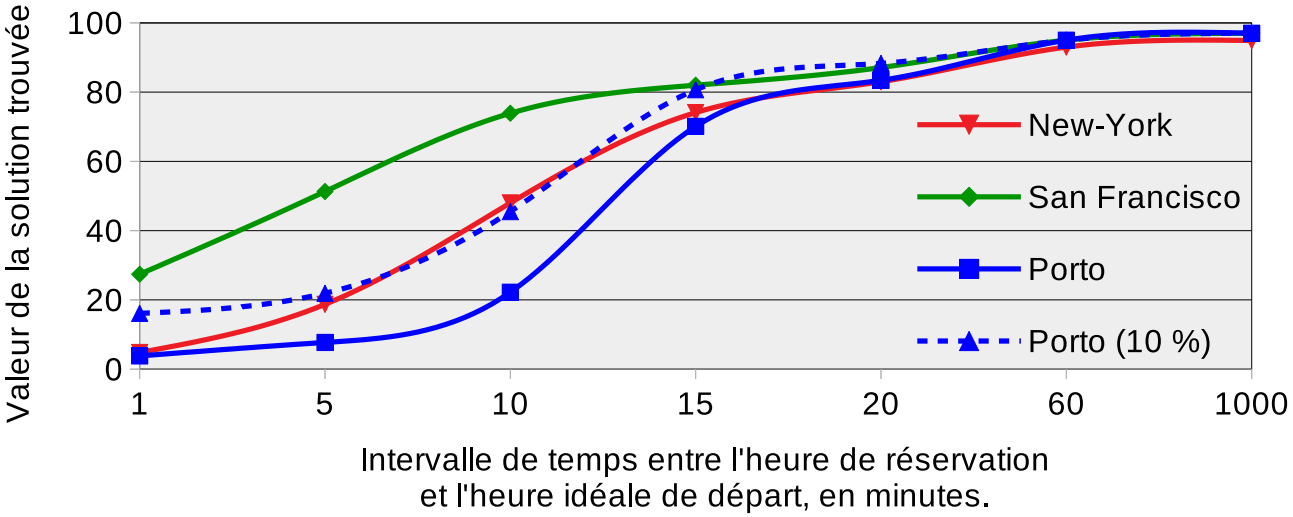


Figure 9.8 – *Valeur des solutions trouvée par l'algorithme glouton en fonction de l'avance avec laquelle les clients réservent. (10%) signifie dans ce tableau "Avec 10% des requêtes ayant été formulées un jour en avance". Les clients acceptent toujours un retard égal à exactement 20% de la durée prévue de leur trajet, ce qui signifie généralement un peu plus de 2 minutes. Il n'y a pas de retard fixe toléré. Les taxis ont une batterie infinie.*

où les taxis ont une batterie infinie, l'algorithme glouton fonctionne bien lorsque les clients acceptent d'être récupérés 20 minutes après qu'ils aient formulé leur requête, comme le montre la figure 9.8.

Sans surprise, lorsque des clients ne tolèrent presque pas de retard et formulent au dernier moment leur requête, l'algorithme glouton est incapable de les traiter. Dans des instances aussi particulières, la qualité de la solution trouvée est vraisemblablement proportionnelle à la capacité de l'algorithme utilisé à prédire où les prochains clients vont apparaître. Ce n'est pas un cas inintéressant à étudier, mais il sort du cadre de notre étude.

9.3.1 Lien entre instant de réservation et recharge

Lorsque l'infrastructure de recharge est restreinte (un chargeur de 18kW pour 5 taxis), l'avance avec laquelle les clients doivent formuler leur requête augmente. Les figures 9.9 et 9.10 présentent la variation du niveau de charge des taxis durant quatre simulations à New-York avec 20 taxis et 6 chargeurs.

Dans le cas statique de la figure 9.9, la valeur de la solution obtenue est proche de celle obtenue avec des taxis ayant une batterie infinie. On observe un niveau moyen de batterie élevé, un cycle jour-nuit en semaine, un plateau le week-end (fin de simulation), et un faible écart entre les quartiles à chaque instant.

Expérimentalement, nous avons observé que, pour que la solution trouvée dans le problème dynamique soit similaire à celle trouvée dans le problème statique, les clients doivent réserver avec une avance k, où k est le temps nécessaire pour une charge complète. Avec des chargeurs de 18kW, on a $k = 4$ heures. Avec des chargeurs de 72kW, on a $k = 1$ heure. Autrement dit, plus les chargeurs sont lents, plus il est intéressant d'inciter les clients à formuler leurs requêtes tôt.

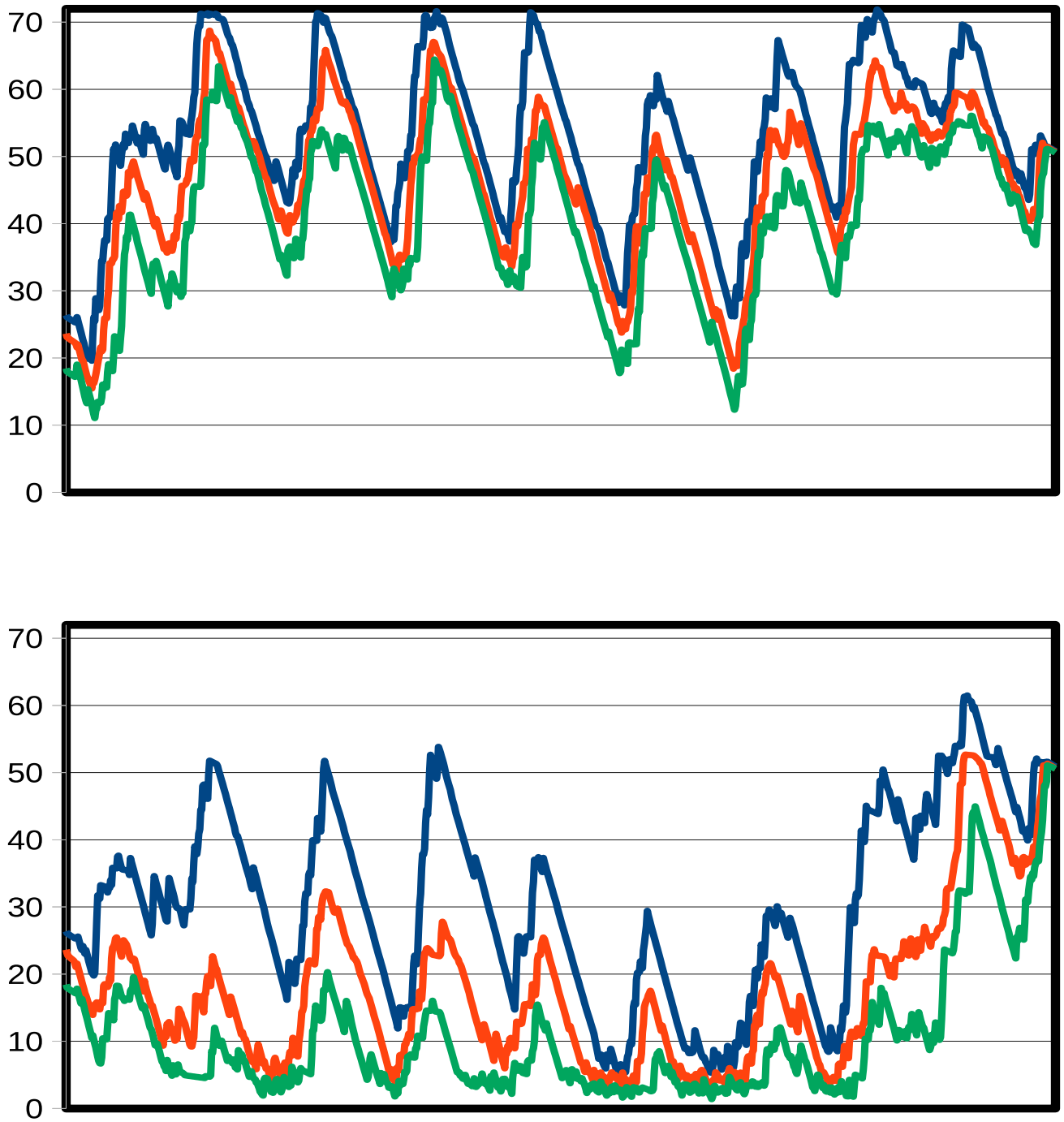


Figure 9.9 – *Ces deux figures représentent le niveau de batterie des taxis (en kWh) au cours de deux simulations. En haut, il s'agit du cas statique avec SLC. En bas, il s'agit du cas dynamique avec formulation 60 minutes en avance. Les courbes présentent les trois quartiles. Autrement dit, à chaque instant, un quart des taxis ont un niveau de batterie sous la courbe verte, la moitie ont un niveau sous la courbe rouge, et les trois quarts ont un niveau sous la courbe bleue.*

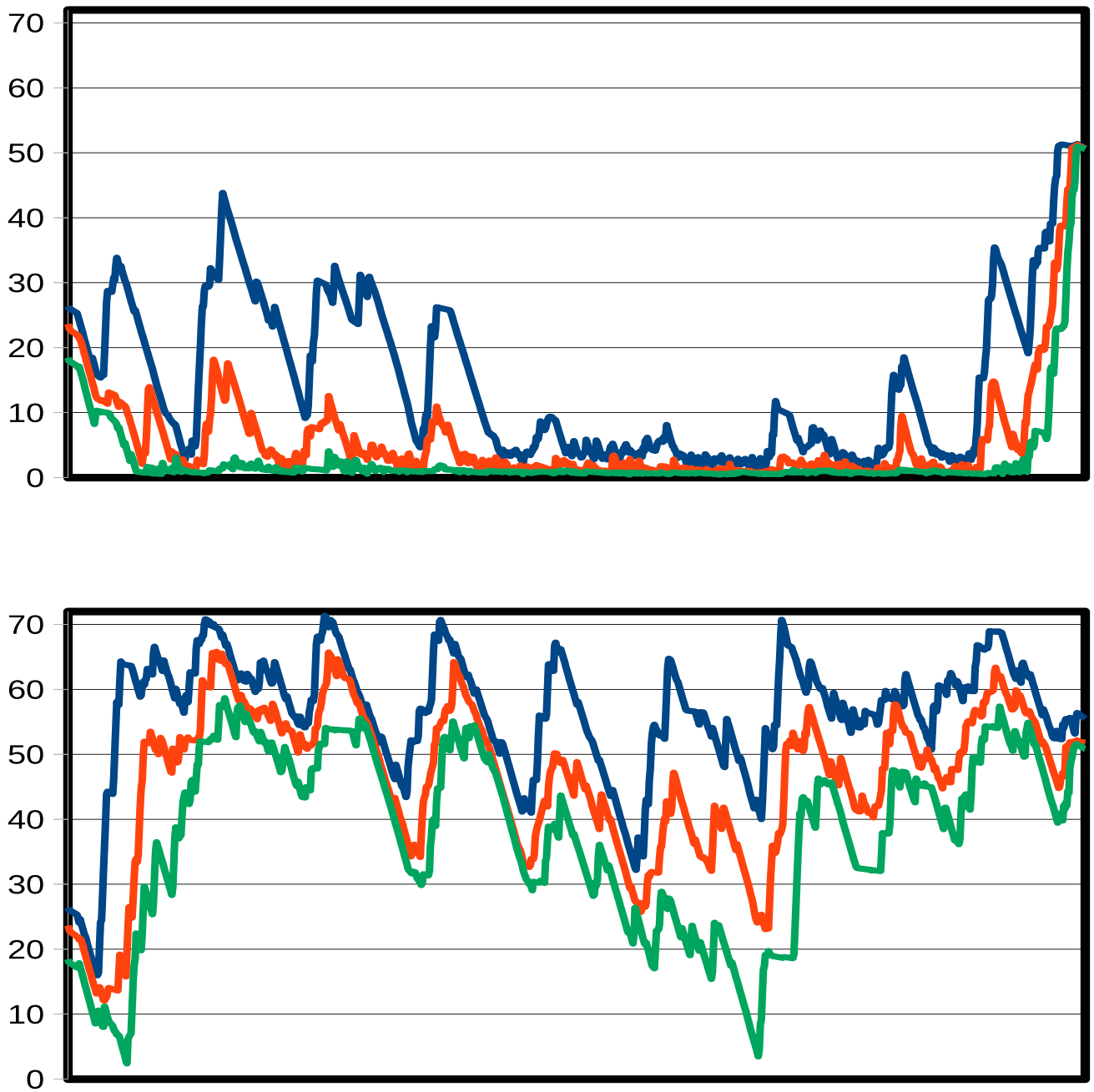


Figure 9.10 – *Quartiles du niveau de batterie des taxis au cours de deux simulations, dans lesquelles les clients réservent 20 minutes en avance. Dans le second graphique, une charge commencée ne peut pas être raccourcie et une charge ne peut pas donner moins de 25% de batterie.*

Toujours dans la la figure 9.9, on peut observer le niveau de charge des taxis lorsque les clients réservent avec une avance inférieure à $k = 4$ heures. Le niveau moyen de batterie est beaucoup plus faible, avec à certains instants des écarts importants entre le niveau de batterie des taxis. L'utilisation de l'infrastructure de recharge est moins efficace que dans le cas statique, avec à la fois plus de charge et beaucoup plus de consommation énergétique. La valeur de la solution est également légèrement plus mauvaise ($\sim 1\%$).

On a constaté dans la sous-section précédente que si les taxis ont une batterie infinie dans une instance avec des réservations 20 minutes en avance, la solution trouvée est proche en qualité d'une solution trouvée avec des réservations 1000 minutes en avances. La figure 9.10 montre que ce n'est plus le cas si les taxis ont une batterie finie. La stratégie de charge est extrêmement inefficace, et la valeur de la solution est 30% plus mauvaise que dans le cas statique. Les charges sont clairement repoussées à la fin de la simulation.

La figure 9.10 montre également qu'interdire les charges donnant moins de 25% de batterie peut forcer les taxis à conserver un niveau de batterie plus élevé. Cela permet d'augmenter la qualité de la solution de 20%. Interdire ces charges courtes a cependant un impact négatif dans le cas statique, diminuant la valeur de la solution de quelques pourcents.

Pour conclure l'étude de l'impact de l'instant de réservation, on peut donc dire que :

— Réserver 20 minutes en avance est suffisant lorsque l'infrastructure de recharge et les batteries des taxis sont surdimensionnées.

— Sinon, il faut que les clients réservent avec suffisamment d'avance pour permettre une charge complète.

— Interdire les charges très courtes est une méthode qui peut être utilisée pour améliorer la solution lorsque les clients réservent avec trop peu d'avance.

9.4 Résultats des recherches de voisinage

Dans cette section, nous présentons les performances du recuit simulé, et nous les comparons à une montée de colline.

9.4.1 Montée de colline

Nous avons implémenté une montée de colline ayant le fonctionnement suivant. A chaque itération, on ne choisit ni le meilleur mouvement (montée de colline *steepest ascent*) ni un mouvement aléatoire (montée de colline stochastique). A la place, on teste un par un tous les mouvements, et on utilise le premier mouvement améliorant trouvé.

5 fonctions de mouvement ont été définies dans la section 7.4 :

— R^{+} : Insertion d'une requête non-traitée, comme dans l'algorithme glouton.

— R^{-} : Retrait d'une requête traitée.

— R^{+-} : Insertion d'une requête non-traitée en retirant un minimum de requêtes déjà traitées. Équivalent à R^{+} si aucun retrait n'est requis.

— R^{swap} : Échange de requêtes entre deux edtt.

— $R^{glouton}$: R^{+-} suivi d'une insertion des requêtes retirées avec l'algorithme glouton.

Une montée de colline avec les fonctions de mouvement R^+ et R^{+-} ne fournit qu'une amélioration négligeable de la qualité de la solution. Dans la suite de ce chapitre, $R^{glouton}$ est le seul mouvement utilisé dans la montée de colline. Sauf mention contraire, l'algorithme glouton contenu dans $R^{glouton}$ est un SLC.

Durée d'exécution

Avant de comparer la qualité des solutions trouvées par la montée de colline aux autres méthodes, présentons ses caractéristiques propres, et en particulier sa durée d'exécution.

On constate d'abord deux choses :

— Une insertion avec $R^{glouton}$ consiste en une exécution de R^{+-}, suivie d'une exécution de R^+ pour chaque requête retirée. En pratique, le temps requis pour une insertion avec $R^{glouton}$ est donc plusieurs fois plus long qu'une insertion avec R^+.

— Une itération de la montée de colline consiste à tenter d'insérer chaque requête qui n'est pas traitée. Dans nos instances, environ 10% des requêtes ne sont pas traitées dans la solution initiale fournie par l'algorithme glouton.

Cela signifie donc concrètement qu'en ordre de grandeur, une itération de la montée de colline dans le pire des cas dure aussi longtemps qu'une exécution complète de l'algorithme glouton.

Sur des instances à San Francisco et Porto avec 35 taxis, il a fallu environ 4 heures pour trouver un optimum local. Sur une instance de New-York avec 35 taxis et des chargeurs de 72kW, il a fallu un peu plus d'une heure à la montée de colline pour trouver un optimum.

Le temps de calcul est largement influencé par la puissance des chargeurs. En ignorant les contraintes de charge dans l'instance de New-York, il a fallu moins de 10 minutes de temps d'exécution pour atteindre un optimum local. A l'inverse, en remplaçant les chargeurs rapides (72kW) par des chargeurs lents (18kW) quatre fois plus nombreux, le temps de calcul passe à près de 16 heures.

Dans des instances avec 25 taxis, le nombre de mouvements améliorants trouvés durant la montée de colline est généralement de l'ordre de quelques centaines, mais il peut dépasser le millier (en particulier si la solution initiale est mauvaise). Parmi les instances dont la solution initiale a été générée par SLC ou VHC, le plus grand nombre d'améliorations a été observé dans une instance avec VHC à San Francisco, avec 11 000 requêtes, et dans laquelle la solution est améliorée 2064 fois durant une montée de plus de 8 heures.

Performances par rapport à l'algorithme glouton

Présentons maintenant l'amélioration permise par la montée de colline par rapport à une solution initiale fournie par l'algorithme glouton. La table suivante présente l'amélioration obtenue sur une instance donnée, en fonction des paramètres de l'algorithme glouton, sur une instance de San Francisco avec 30 taxis et 6 chargeurs.

algorithme glouton	valeur de la solution initiale	valeur de la solution finale
R-R	83.1	94.1
VHC	95.3	97.6
SLC	96.5	98.4

Dans nos expérimentations, nous avons systématiquement constaté trois choses, que l'on retrouve dans le tableau précédent :

— La montée de colline améliore toujours la solution trouvée par l'algorithme glouton.

— Plus la solution initiale est loin de l'optimale, plus l'amélioration est importante.

— La meilleure stratégie sans montée de colline (SLC ici) est aussi la meilleure stratégie avec une montée de colline.

Ces observations sont valides indépendamment de la ville étudiée, du nombre de taxis ou de chargeurs.

En réduisant le nombre de chargeurs ou le nombre de taxis de sorte que seules 75% des requêtes sont traitées avec SLC, la montée de colline augmente la valeur de la solution de plus de 10 points. Elle est donc capable d'améliorer la solution quelle que soit la raison pour laquelle la solution trouvée par l'algorithme glouton est mauvaise.

9.4.2 Recuit simulé

Le recuit simulé est une méthode qui requiert un ajustement manuel de plusieurs de ses paramètres, et les bons paramètres sont établis empiriquement pour un type d'instance donné. Rappelons ici les paramètres de notre recuit et les valeurs qui ont été considérées.

Paramètre	Valeur
Solution initiale	Fournie par l'algorithme glouton
Température initiale et finale	entre 0 et 10000
Fonction d'évolution de la température	Ces fonctions sont définies dans la section 7.3.1. Il en existe quatre : linéaire, linéaire avec réinitialisation périodique, géométrique, et sinusoïdale.
Limite de temps	quelques dizaines de minutes ou quelques heures
Un ensemble de fonctions de mouvement, et un vecteur de probabilité pour en choisir une à chaque itération	Les fonctions de mouvement sont celles de la section 7.4

Correctement configuré, le recuit est capable d'améliorer significativement une solution, comme le montre la figure 9.11. Dans certains cas, une amélioration de près de 10 points à lieu, comparé à la solution trouvée par l'algorithme glouton.

Les résultats de la figure 9.11 ont été obtenus avec les paramètres suivants du recuit. La probabilité à chaque itération d'utiliser $R^{glouton}$ est 50%, celle d'utiliser R^{-} est 10%, et celle d'utiliser R^{swap} est 40%, et le glouton exécuté dans $R^{glouton}$ est un SLC. La température initiale est 500, la température finale est 0, avec une décroissance linéaire. L'exécution du recuit dure 15 minutes. L'algorithme glouton inclus dans $R^{glouton}$ est un SLC; utiliser une autre stratégie a un effet négligeable sur la qualité de la solution (inférieure à 0.05).

Dans cette figure, on peut constater que, avec un recuit de 15 minutes, plus la valeur de la solution initiale est bonne, plus la valeur de la solution finale est bonne.

Le tableau 9.2 présente la valeur de la solution trouvée avec un recuit aux paramètres similaires à ceux utilisés dans la figure 9.11. Le principal enseignement de ce tableau est que

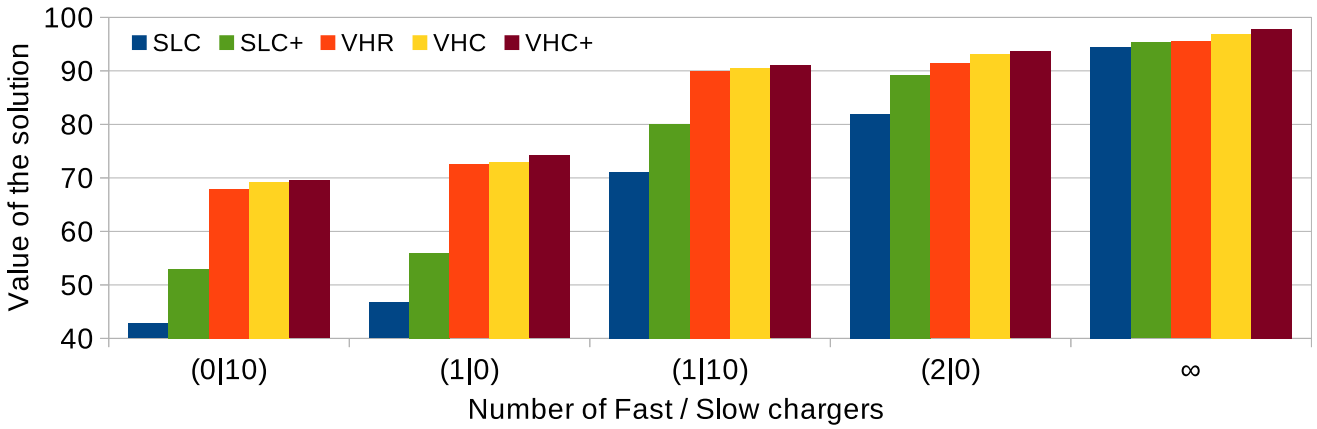


Figure 9.11 – *Valeur de la solution trouvée par l'algorithme glouton sur différentes instances. Un '+' signifie que la solution a été améliorée par un recuit simulé. Les caractéristiques de ces instances sont décrites dans [Hoché et al., 2020]. Il y a 35 taxis. VHR est ici donné à titre indicatif.*

Temps (minutes)	Valeur (points)	amélioration (points)
0	94.50	
3	94.99	+0.49
10	95.50	+0.51
25	95.98	+0.48
60	96.22	+0.24
180	96.36	+0.14

Table 9.2 – *Valeur d'une solution trouvée avec un recuit, en fonction du temps de calcul. L'amélioration est exprimée par rapport à la valeur de la solution de la ligne précédente.*

Temps (minutes)	Valeur (Recuit)	Valeur (Montée de colline)	différence
0	94.50	94.50	0
3	94.99	95.12	0.13
10	95.50	95.77	0.27
25	95.98	96.33	0.35
60	96.22	96.64	0.42
180	96.36	96.67	0.31

Table 9.3 – *Valeur d'une solution trouvée avec un recuit, en fonction du temps de calcul.*

plus de la moitié des améliorations ont lieu durant les 10 premières minutes d'exécution, et 80% ont lieu durant les 25 premières minutes.

A partir du tableau 9.2 et de la figure 9.11, on peut constater que plus la valeur de la solution initiale est bonne, plus la valeur de la solution finale est bonne, sauf en ayant un recuit d'une durée extrêmement élevée. L'explication la plus probable est que la durée d'exécution de $R^{glouton}$ (quelques dizièmes de secondes sur nos instances) est trop élevée pour permettre au recuit simulé de s'écarter beaucoup de la solution initiale.

Présentons maintenant les résultats obtenus avec les autres fonctions de mouvement. Dans le cas d'un recuit qui utilise le mouvement R^+ avec une probabilité p et le mouvement R^- avec une probabilité $1-p$, les résultats sont très mauvais. Un tel recuit peut produire quelques améliorations durant la première minute, mais après cela, la recherche gravite systématiquement vers des solutions sensiblement plus mauvaises que la solution initiale, ou bien la température est tellement faible que l'écrasante majorité des mouvements sont rejetés.

Cela est probablement dû à la forme du graphe des voisinages. Puisqu'il n'est pas permis de passer par une solution inadmissible, et puisqu'il est plus facile de retirer une requête que d'en insérer une, il est difficile de rester dans une région avec un nombre très élevé de requêtes par edtt, mais la température envoie vers de telles solutions. Il y a donc sans doute un équilibre vers lequel le recuit converge.

Le constat reste le même en remplaçant R^+ par R^{+-}. Le mouvement $R^{glouton}$ est clairement le meilleur. Les mouvements R^{swap} et R^- peuvent le complémenter, mais ne semblent pas avoir d'impact significatif sur la solution.

Concernant l'évolution de la température, il ne semble pas y avoir de fonction offrant de résultats sensiblement meilleures qu'une autre, lorsque la solution initiale est fournie par l'algorithme glouton.

Comparaison avec la montée de colline

La montée de colline et le recuit simulé utilisant la même fonction de mouvement $R^{glouton}$, il était attendu que, à temps de calcul égal, la montée de colline fournisse une meilleure solution que le recuit simulé. Cette hypothèse s'est avéré correcte dans notre cas, comme le montre la table 9.3, qui présente les résultats obtenus par une montée de colline sur l'instance utilisée dans la table 9.2, en fonction de la durée d'exécution.

A partir de ce tableau, on peut supposer que, pour que le recuit trouve une solution meilleure que la montée de colline sur cette instance, sa durée d'exécution doit être bien plus longue, se comptant éventuellement en jours.

Instance	Algorithme glouton	Montée de colline	Recuit Simulé
1	74.65	+4.28 (78.93)	+0.98 (79.91)
2	75.03	+4.41 (79.44)	+1.35 (80.79)
3	88.99	+ 2.70 (91.69)	+ 0.52 (92.21)

Table 9.4 – *Valeur de la solution trouvée par l'algorithme glouton, améliorée avec une montée de colline puis un recuit d'une heure. La ville est New-York. Dans l'instance 1, il y a 5 taxis, 2500 requêtes, 1 chargeur de 18kW et 2 chargeurs de 7.2kW. L'instance 2 est comme l'instance 1, mais les taxis ont une batterie infinie. Dans l'instance 3, il y a 20 taxis, 4 chargeurs de 18kW, et 8600 requêtes.*

Pour cette raison, pour obtenir les meilleurs résultats avec le temps de calcul le plus faible, il parait judicieux de ne pas commencer un recuit simulé à partir d'une solution fournie par l'algorithme glouton, mais à partir d'une solution fournie par la montée de colline. Le tableau 9.4 présente l'amélioration obtenue avec une telle méthode sur trois instances.

Dans les instances du tableau 9.4, le recuit simulé est différent de celui utilisé précédemment. Plutôt qu'une descente de température linéaire avec une température initiale de 500, on utilise une descente sinusoïdale (*cf.* section 7.3.1) avec une température initiale de 700. Le sinusoïde a une période de 2000 mouvements ; avec 20 taxis, cela signifie qu'une température de 0 est atteinte tous les quarts d'heure.

Cette stratégie est plus efficace que les stratégies ne remontant jamais la température. Ces dernières ne permettent qu'une augmentation marginale de la qualité de la solution, au mieux de l'ordre de 0.05 points sur des instances avec au moins 20 taxis.

Utiliser une descente linéaire avec ré-initialisation est légèrement moins efficace que la descente sinusoïdale, lorsque la durée d'exécution du recuit simulé est d'une heure. Nous avons testé ces deux stratégies dans 10 instances ayant les mêmes paramètres que l'instance 1 du tableau 9.4, mais une initialisation différente du générateur de nombres aléatoires. Dans ces instances, la descente sinusoïdale a permis une amélioration moyenne de 1.14 points et un écart type de 0.37 points, tandis que l'autre stratégie a amélioré de 0.88 points avec un écart type de 0.32 points.

Notons cependant qu'il est incertain que la descente sinusoïdale soit la meilleure stratégie dans le cadre d'un recuit de plusieurs heures voire plusieurs jours. Une autre hypothèse qui mériterait d'être vérifiée est que, pour un recuit de moins d'une heure, une alternance binaire de la température est plus efficace. Plus précisément, alterner phase d'exploration à 700 degrés et phase d'exploitation à 0 degré, sans jamais passer par des températures comprises entre 1 et 699. Cette méthode s'approcherait d'une LNS (Large Neighbourhood Search [Parragh and Schmid, 2013]).

Nous n'avons pas étudié ces hypothèses plus en détail, pour nous concentrer sur d'autres questions relatives aux instances, comme celles de la prochaine section.

9.5 Observations relatives aux instances

Avant de clore ce chapitre, présentons des solutions obtenues à l'aide de l'algorithme glouton, et ce que ces solutions nous apprennent sur les instances que nous avons implémentées.

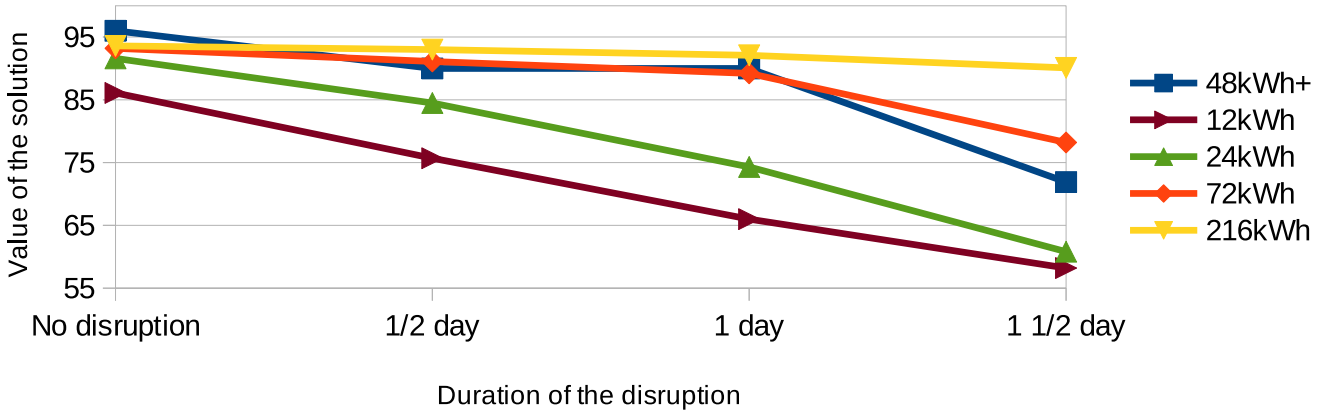


Figure 9.12 – *Valeur de la solution trouvée par l'algorithme glouton à Porto, face à un arrêt complet de l'infrastructure de recharge 24 heures après t_0. Chaque courbe est associée à une capacité de batterie des taxis. Pour la courbe "48kWh+", l'infrastructure de recharge contient 50% plus de chargeurs.*

9.5.1 Partage de course

Dans des instances dans lesquelles au moins 9000 requêtes sur 10000 sont traitées, environ 40% des usagers voient leur course partagée, que ce soit à Porto, New-York, ou San-Francisco.

Un facteur ayant un impact majeur sur le taux de partage est la densité des requêtes. Plus le nombre de requêtes par heure par kilomètre est élevé, plus le taux de partage est élevé. En multipliant par 10 le nombre de requêtes et le nombre de taxis, le taux de partage passe à 66%. En multipliant par 5 le nombre de requêtes, avec 15 places par taxis et une heure de retard toléré, le taux de partage passe à 90%.

D'autres facteurs ont une importance plus mineure. Une faible capacité de batterie pour les taxis diminue par exemple le taux de partage.

9.5.2 Disponibilité de l'infrastructure de recharge

Dans cette sous-section, nous présentons la taille idéale pour la batterie des taxis, en étudiant en particulier l'impact de la disponibilité des chargeurs. Ces résultats ont aussi été présentés dans [Hoché et al., 2020].

Comme il a été mentionné dans la section 3.1.3, la disponibilité des chargeurs est limitée de trois manières. 1) Un chargeur ne peut être utilisé que par un taxi à la fois, 2) Les chargeurs sont parfois inutilisables, et 3) il faut patienter pour accéder à un chargeur, pour manœuvrer et faire la queue.

Dans une instance, nous avons évalué la résilience de la flotte face à une disruption, et plus précisement, un arrêt général de l'infrastructure de recharge. Le résultat obtenu est résumé dans la figure 9.12. Dans cette instance, la ville est Porto, les taxis peuvent bouger pendant 4 jours, en commençant un jeudi.

Deux points sont intéressants concernant la figure 9.12. Premièrement, on constate que la qualité de la solution trouvée sans disruption dépend très peu de la taille des batteries. Des taxis avec seulement 24 kWh de batterie ont presque la même efficacité que des taxis avec 216kWh.

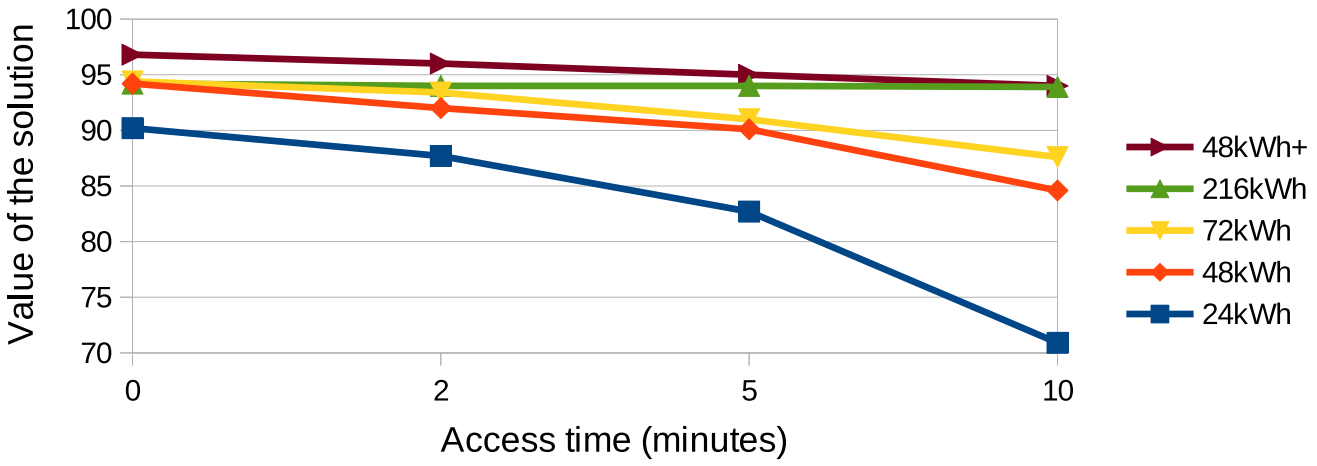


Figure 9.13 – *Qualité de la solution trouvée en fonction du temps nécessaire pour accéder à un chargeur. Chaque courbe est associée à une capacité de batterie des taxis. Pour la courbe "48kWh+", l'infrastructure de recharge contient 50% plus de chargeur.*

Deuxièmement, on constate dans la figure 9.12 qu'une batterie de 72kWh est suffisante pour supporter une disruption de 24 heures.

Dans une seconde instance, nous avons évalué les performances de la flotte en fonction du temps nécessaire pour accéder à un chargeur. Les résultats sont présentés dans la figure 9.13.

La figure 9.13 permet de conclure à la supériorité d'une infrastructure de recharge publique, en comparant la courbe rouge et la courbe mauve. Entre $n \in \mathbb{N}$ chargeurs privés, auxquels les taxis peuvent se connecter instantanément, et $1.5n$ chargeurs partagés pour lesquels 10 minutes d'attente sont nécessaires à chaque connexion, il apparaît clairement que cette seconde voie est plus intéressante. Ce résultat suppose cependant que le temps d'attente est connu pour des chargeurs publics, il serait donc intéressant de réitérer cette comparaison dans un modèle stochastique.

La figure 9.13 montre également que, sans surprise, plus les taxis ont une capacité de batterie limitée, plus ils sont pénalisés lorsqu'il existe un d'attente fixe avant les charges.

Pour conclure l'étude de la disponibilité des chargeurs, on peut donc dire que :

— Dans un modèle déterministe, en régime de croisière, des tailles de batteries faibles ($\sim$24kWh) sont suffisantes.

— Des véhicules milieu-de-gamme de 2020, avec une batterie de 72kWh sont capables de faire face à des arrêts de 24 heures de l'infrastructure de recharge, à condition que la disruption soit connue 24 heures en avance.

— Avec une infrastructure de recharge ayant une file d'attente importante ($\sim$10 minutes), des grosses batteries sont préférables ($200+$ kWh).

— Il est préférable de partager l'infrastructure de recharge pour avoir un ensemble de chargeurs le plus grand possible, même si cela implique une augmentation significative du temps d'attente avant chaque charge.

9.6 Conclusion sur les méthodes de résolution

Dressons maintenant un bilan des résultats présentés dans ce chapitre, concernant l'algorithme glouton pour le problème statique et le dynamique, ainsi que sur les méthodes d'amélioration des résultats.

Dans le problème statique, lorsque la recharge est peu contrainte, l'algorithme glouton présente de bons résultats avec SLC, qui consiste à insérer les requêtes par ordre chronologique, et à envoyer à chaque fois le taxi le plus proche lorsque plusieurs taxis peuvent traiter un client. SLC est proche de CLC (tri par instant de réservation / Call time), qui est la stratégie historiquement utilisée par les taxis. Cette stratégie montre cependant ses faiblesses lorsque la charge est particulièrement contrainte.

La stratégie VHC, qui consiste à insérer dans la solution les requêtes de plus forte valeur en priorité présente des résultats qui sont à la fois bons et stables. La figure 9.2 montre cependant que VH- donne une solution 4 à 5 points moins bonne que SL- à New-York, vraisemblablement à cause d'une difficulté à permettre le partage de course.

La stratégie –M qui consiste à envoyer le taxi avec le plus de batterie présente parfois les meilleurs résultats, mais fonctionne plutôt mal lorsque la recharge est peu contrainte, avec des performances similaires à –R.

Dans le problème statique, pour trouver la meilleure solution dans le temps le plus court, la meilleure approche consiste à générer une solution avec l'algorithme glouton, l'améliorer avec une montée de colline, et suivre avec un recuit simulé. La montée de colline et le recuit utilisent tous les deux le mouvement $R^{glouton}$. Si le temps de calcul permis pour le recuit est de quelques heures seulement, une gestion de température avec des remontées régulières de température est à privilégier, en particulier, la descente de température sinusoïdale.

Comme nous l'avons souligné dans le chapitre 5, la taille des instances que nous avons considérées rendent complètement impossible l'application d'une méthode exacte de résolution. Le constat est moins négatif concernant les méta-heuristiques, mais cela constitue néanmoins un élément limitatif pour leur application. Quand une simple montée de colline prend plus de dix heures, on comprend bien qu'il va être difficile d'exploiter une méta-heuristique à sa pleine puissance.

De plus, il ne paraît guère envisageable de considérer des mouvements plus diversificateurs, mais plus gourmands en temps de calcul, afin d'augmenter la connexité du graphe des voisinages, ou bien de considérer des méta-heuristiques qui, à chaque itération, énumère toutes les solutions voisines.

Concernant la méthode dynamique, le résultat le plus intéressant est sans doute la figure 9.7, qui montre le caractère court-termiste de SLC, qui est proche de celui de CLC (problème dynamique). Interdire aux taxis de traiter des clients supplémentaires lorsque le niveau moyen d'énergie de la flotte passe en dessous d'un certain seuil pourrait être envisagé. Simplement envoyer le taxi avec le plus ou le moins de batterie ne semble en tout cas pas satisfaisant.

L'algorithme glouton est largement suffisamment scalable pour une utilisation en temps réel, même dans des instances avec des centaines de milliers de requêtes, à condition d'avoir un nombre suffisant de processeurs. La montée de colline et le recuit simulé sont envisageables pour améliorer des instances avec des dizaines de milliers de requêtes.

Chapitre 10

Conclusion et perspectives

Comme mentionné dans l'introduction générale, l'objectif de cette thèse était de mettre au point un service gérant une flotte d'une centaine de taxis électriques autonomes et partagés, de manière centralisée, en prenant en compte une infrastructure de recharge restreinte, avec des chargeurs partagés à la fois entre les taxis et avec d'autres véhicules particuliers. Plus spécifiquement, l'objectif était d'assurer que la gestion des charges ne perturbe pas le service.

Synthèse des contributions

Nous avons proposé la définition d'un nouveau problème, le PCE-ADARP, une modélisation pour celui-ci, ainsi qu'un processus complet permettant, à partir de données libres et de logiciels ouverts, de générer des instances de ce problème. Ces instances sont réalistes, comparées aux traditionnelles instances de Solomon.

Nous avons évalué les limites des méthodes exactes. Nous avons établi que le temps de calcul requis par des solveurs industriels (comme CPLEX) pour résoudre une instance du PCE-ADARP évolue exponentiellement avec le nombre de requêtes. Une approche envisageable concernant les méthodes exactes est l'utilisation de la programmation par contrainte, qui est potentiellement plus adaptée qu'une résolution de type branch-and-bound, compte tenu des nombreuses contraintes du problème.

Nous avons proposé un algorithme glouton dans lequel l'insertion de requêtes et l'insertion de charges sont traitées séparément. Nous avons montré que chacun de ces deux sous-problèmes est NP-complet. L'algorithme glouton est scalable pour des instances ayant des réservations dix jours en avance, des centaines de taxis, et des dizaines voire centaines de milliers de requêtes. Nous avons aussi rendu cet algorithme parallélisable. De plus, il ne commet pas d'erreur évidente sur des instances réalistes.

Nous avons montré que la stratégie à employer lors de l'exécution de l'algorithme glouton doit être adaptée à la situation, en particulier la disponibilité et puissance de l'infrastructure de recharge, et la facilité du partage de course dans la région considérée. Cela est plus vrai dans le cas statique, compte tenu de l'importance de l'ordre d'insertion des requêtes. Les stratégies qui se sont distinguées consistent à envoyer le taxi le plus proche ou avec le plus de batterie, et d'insérer les requêtes soit par ordre chronologique soit par valeur décroissante.

Dans le problème dynamique, nous avons établi un ensemble de situation posant des

difficultés pour l'algorithme glouton à l'aide d'une comparaison avec les résultats obtenus pour le problème statique, et des explications pour ces difficultés.

Nous avons implémenté un algorithme de montée de colline et un recuit simulé. Nous avons déterminé les limitations théoriques, notamment, la non-connexité du graphe des voisinages. En limitant le temps de calculs à quelques dizaines de minutes ou quelques heures, et sur des instances comptant au moins des milliers de requêtes, la meilleure approche consiste à utiliser l'algorithme glouton, puis la montée de colline, puis enfin le recuit simulé avec une descente de température sinusoïdale. Sur nos instances, chaque méthode a permis d'améliorer sensiblement les résultats fournis par la méthode précédente. Compte tenu de la taille de nos instances et de l'impact que cela a sur la durée d'exécution de nos heuristiques, il n'est pas certain que la non-connexité du graphe des voisinages pose problème dans le cadre d'un recuit durant seulement quelques heures.

Enfin, nous avons étudié l'impact des caractéristiques des instances comme le degré d'avance avec lequel les clients réservent, la disponibilité de l'infrastructure de recharge, ou le taux de partage. Sans recharges requises, une faible visibilité sur les futures requêtes pose peu de difficulté, et influence peu la qualité de la solution. Avec recharges requises, les réservations doivent être beaucoup plus précoces. Avec des chargeurs accessibles rapidement et tout le temps, de petites batteries de taxis suffisent. Cependant, de telles flottes de taxis sont fragiles face à des disruptions de l'infrastructure de recharge, et supportent assez mal les infrastructures partagées lorsque les files d'attentes sont longues. Des batteries de 72kW, proches de celles des véhicules électriques milieu-de-gamme de 2020 semblent adaptées.

Perspectives

Afin de faciliter la recherche dans le domaine du transport, il paraît important qu'un simulateur de transport de personne d'échelle macroscopique soit mis en place. Cela permettrait d'une part de réduire le temps passé à développer des outils de simulation (une part importante de cette thèse), et cela permettrait d'autre part de faciliter les comparaisons et vérifications des résultats obtenus par des pairs. Il serait particulièrement intéressant que ce simulateur soit open-source, multi-modal, et contenant une interface avec divers jeux de données libres, dont en particulier OSM.

Concernant le modèle, de nombreuses améliorations pourraient être apportées pour le rendre plus réaliste, mais deux aspects en particulier semblent intéressants à développer. Premièrement, ajouter une stochasticité et incertitude dans la disponibilité des chargeurs. Deuxièmement, développer la recherche de stationnement, que nous avons modélisé très simplement. La recherche scientifique relative à ces deux points est actuellement limitée par un manque de données de qualité sur le sujet.

Dans le problème dynamique, l'absence de prédictions est surtout un problème lorsque l'infrastructure de recharge est contrainte. Dans ce type d'instance, il serait intéressant de comparer les résultats obtenus avec une méthode insérant d'abord des charges puis des requêtes, plutôt que l'inverse, ou d'incorporer un système de prédiction des futures requêtes.

Dans le problème statique, avant de réfléchir à des méthodes plus complexes pour trouver des meilleures solutions, la priorité paraît être de diminuer le temps de calcul. La complexité élevée de l'ajout de charge, couplée au fait que l'algorithme d'ajout de charge est utilisé après chaque mouvement a un impact majeur sur le temps de calcul. Utiliser des mé-

canismes permettant d'avoir à n'opérer des vérifications et modifications d'emploi du temps que localement est une possibilité, ou un ajout préemptif de charges longues, pour ne pas avoir à ajuster en permanence la quantité de recharges.

Plus généralement, dans des instances dans lesquelles un aspect du problème est particulièrement prégnant (partage de course, indisponibilité des chargeurs, ...), il pourrait être intéressant de comparer nos heuristiques, relativement généralistes, à des heuristiques spécialisées, qui seraient sans doute plus mauvaises dans le cas général mais meilleures pour ces instances. Pour le partage de course par exemple, insérer les requêtes non pas une-par-une, mais par paquets, en regroupant les requêtes ayant à peu près la même origine et destination.

Enfin, pour le recuit simulé, une gestion binaire de la température devrait être évaluée, et une étude des performances des différentes descentes de température sur un temps plus long pourrait également fournir des résultats intéressants.

Conclusion finale

De nombreuses problématiques ont été jointes dans cette thèse afin d'établir une vision holistique d'un problème, et trouver des méthodes de résolution applicable à des cas réels de grande taille. Les résultats obtenus peuvent être renforcés ou étendus de nombreuses manières. Nous espérons que les travaux réalisés durant cette thèse pourront servir de ressource pour des travaux additionnels plus ciblés, qui seraient ainsi très complémentaires.

Annexe A

Formulation en MILP du problème

Cet annexe présente la description du modèle du chapitre 5, qui comporte quelques simplifications et différences comparée au modèle du chapitre 4.

Ainsi, la durée d'une connexion à un chargeur est égale à la durée d'une recherche de place de stationnement. La durée des trajets ne dépend pas de l'instant de départ. Les taxis doivent finir avec une batterie non-vide. Chaque arc a une consommation énergétique de base, plutôt qu'une longueur et une durée. Les taxis n'ont pas de consommation par mètre, et ils commencent avec une batterie pleine.

Le graphe manipulé dans le modèle MILP est le graphe des clients qui est presque le graphe des actions. Il s'agit d'une entrée du problème.

Il n'y a pas de concept d'action ou d'activité, les contraintes sont exprimées directement par rapport au graphe des clients.

Ce MILP a été implémenté en langage OPL (Optimisation Programming Language), et nous l'avons résolu avec IBM ILOG CPLEX 12.9.0.0. L'ordinateur utilisé est un DELL Latitude 7490.

Le langage OPL est très proche du langage mathématique, donc la présentation du modèle ci-dessous est une description pratiquement exacte du code employé. Cependant, CPLEX est capable de transformer des contraintes non-linéaires en contraintes linéaires, ce qui permet à la fois de simplifier la modélisation et permet à CPLEX de faire des optimisations supplémentaires. Ces cas sont détaillés ici.

A.1 Entrées

La table A.1 résume les variables qui composent chaque taxi, requête et chargeur. Ces variables sont toutes positives strictes.

$taxi, request$ et $charger$ désignent l'ensemble des taxis, requêtes et chargeurs, respectivement.

$G = (V, A)$ est un graphe orienté. $V = \{Beginning, Pickup, Dropoff, Charge, depot\}$. Un arc a une durée $t \in \mathbb{N}$ et une consommation énergétique de base $w \in \mathbb{R}$. Les notations $a_{(u,v)}$ et (u, v) désignent l'arc allant du sommet u au sommet v.

Tuple	Variable	domaine	représente
Taxi	q	$\mathbb{N}$	nombre de place
	$battery$	$\mathbb{R}$	taille de batterie
	$multiplier$	$\mathbb{R}$	vitesse de consommation énergétique
Requête	α	$\mathbb{N}$	Heure de départ idéal
	ω_1	$\mathbb{N}$	Heure de départ au plus tard
	ω_2	$\mathbb{N}$	Heure d'arrivée au plus tard
	q	$\mathbb{N}$	nombre de passagers
Chargeur	w	$\mathbb{N}$	Puissance

Table A.1 – *Données d'entrées du problème.*

G est un graphe complet. Certains arcs ont une durée infinie, ce qui les rend inutilisables. Ces arcs sont ceux allant vers les sommets de $Beginning$, partant de $depot$, allant de $Pickup$ vers $depot$ ou $Charge$, et enfin allant de $Charge$ vers $Dropoff$.

Les taxis peuvent bouger à l'instant t_0 et doivent être à $depot$ à l'instant t_{end}. Le taxi $taxi_k$ commence au sommet $beginning_k \in Beginning$. Les arcs vers le dépôt ont une durée et une consommation nulle.

On note $p_r \in Pickup$ l'origine de la requête r et $d_r \in Dropoff$ sa destination.

Chaque chargeur peut être utilisé $uses$ fois. $charge^i_j$ désigne la i^{me} charge au chargeur $charger_j$, elle est associée au sommet $charge_{j \times uses+i} \in Charge$.

$searchmax \in \mathbb{N}$ désigne le temps de recherche maximal d'une place de parking ou d'un chargeur.

A.2 Variables de décision

Une solution est définie comme une attribution de valeur aux variables suivantes :

$x^t_a \in \{0, 1\}$ indique si le taxi t utilise l'arc a.

$arrive_v \in \mathbb{N}^*$ instant d'arrivée dans le sommet v

$depart_v \in \mathbb{N}^*$ instant de départ du sommet v

$search_v \in \mathbb{N}^*$ durée de la recherche d'une place dans v

$q_v \in \mathbb{N}^*$ Nombre de passagers au départ du sommet v

$w_v \in \mathbb{R}^+$ Batterie du taxi au départ du sommet v

Dans la suite de cet annexe, et puisque certaines variables ont un nom composé de plusieurs lettres, l'opérateur de multiplication est $\times$ et la notation $taxi_k.q$ désigne la composante q d'un taxi $taxi_k$.
La fonction objective est :

$$\sum_{r \in request,\ t \in taxi} r.q \times a_{(p_r,d_r)}.t \times \sum_{v \in V} x^t_{(p_r,v)} \tag{A.1}$$

A.3 Contraintes

A.3.1 Flot

Ces contraintes servent à s'assurer que la solution ne contient que des chemins élémentaires. Le nombre d'arc sortant maximal est toujours 1 (A.2). Pour chaque sommet, le nombre d'arcs sortants utilisé est identique au nombre d'arcs entrants utilisés (A.3), avec deux exceptions. Pour les sommets de départ des taxis, exactement un arc sortant est utilisé, et par le taxi associé au sommet (A.4). Pour le dépôt, il y a autant d'arcs entrants utilisés que de taxis (A.5) ; A noter que les chemins ne sont pas parfaitement élémentaires puisqu'ils se rejoignent au dépôt.

$$\forall\, u, v \in V \qquad \sum_{t \in taxi} x^t_{(u,v)} \leq 1 \tag{A.2}$$

$$\forall\, u \in V \setminus \{Beginning, depot\}, t \in taxi \qquad \sum_{v \in V} x^t_{(u,v)} - x^t_{(v,u)} = 0 \tag{A.3}$$

$$\forall\, beginning_k \in Beginning \qquad \sum_{v \in V} x^{taxi_k}_{(beginning_k,v)} = 1 \tag{A.4}$$

$$\sum_{u \in V, t \in taxi} x^t_{(u,depot)} = |taxi| \tag{A.5}$$

De plus, le taxi qui récupère un client doit aussi être celui qui le dépose (A.6).

$$\forall\, r \in Request, t \in Taxi, \qquad \sum_{v \in V} x^t_{(p_r,v)} = \sum_{v \in V} x^t_{(d_r,v)} \tag{A.6}$$

A.3.2 Capacité

Les taxis sont vides à t_0 et lors des charges (A.7).

$$\forall\, u \in Beginning \cup Charge \cup depot \qquad q_u = 0 \tag{A.7}$$

La constante M a une valeur qu'on peut considérer comme infiniment élevée. Elle peut être utilisée de diverses manière, notamment pour exprimer un "si". Par exemple, avec $c \in \{0, 1\}$, la phrase "$a < b$ si $c = 1$" peut être écrite "$a < b + M \times (1 - c)$". De la même manière, la phrase "$a = b$ si $c = 1$" peut être écrite "$(a \leq b + M \times (1 - c)) \wedge (a \geq b - M \times (1 - c))$". Ce second cas est utilisé par exemple dans les équations A.9 et A.10.

Les taxis sont vides à t_{end} et avant les charges (A.8).

$$\forall\, u \in V, v \in depot \cup Charge, t \in Taxi \qquad q_u \leq M \times (1 - x^t_{(u,v)}) \tag{A.8}$$

Les taxis se remplissent lors des récupérations de clients (A.9), se vide lors du dépôt de client (A.10).

$$\forall r \in Request, t \in Taxi, u \in V \begin{cases} q_{p_r} \geq q_u + r.q - M \times (1 - x^t_{(u,p_r)}) \\ q_{p_r} \leq q_u + r.q + M \times (1 - x^t_{(u,p_r)}) \end{cases} \quad (A.9)$$

$$\forall r \in Request, t \in Taxi, u \in V \begin{cases} q_{d_r} \geq q_u + r.q - M \times (1 - x^t_{(u,d_r)}) \\ q_{d_r} \leq q_u + r.q + M \times (1 - x^t_{(u,d_r)}) \end{cases} \quad (A.10)$$

A noter que dans le langage OPL (compatible avec CPLEX), l'opérateur "=>" peut être utilisé pour signifier une implication. Le système d'équations A.9 peut ainsi être remplacé par $x_{(u,p_r)} => q_{p_r} == q_u + r.q$. Au delà du gain de visibilité, cette modification permet à CPLEX d'être plus performant. Bien que M puisse être considéré comme infini, les solveurs ont en pratique plus de facilité à résoudre les programmes linéaires traitant de petits nombre (notamment pour des questions d'arrondi) donc utiliser l'opérateur "=>" permet à CPLEX de déterminer automatiquement une bonne valeur pour M, et lui permet aussi de repérer plus facilement des branchements pertinents lors de la recherche de solution.

Tous les systèmes d'équations présentés dans ce chapitre et utilisant un M peuvent être réécris en une seule ligne ne contenant pas de M, en restant utilisable par CPLEX (mais en n'étant plus un système linéaire).

La capacité des taxis ne peut pas être dépassée (A.11).

$$\forall\, u, v \in V, t \in Taxi \qquad q_v <= t.q + M \times (1 - x^t_{(u,v)}) \quad (A.11)$$

A.3.3 Temps

Les taxis commencent à l'instant t_0 (A.12), s'arrêtent à l'instant t_{end} (A.13), et arrivent avant de partir (A.14).

$$\forall\, b \in Beginning \qquad arrive_b = t_0 \quad (A.12)$$

$$\forall\, v \in V \setminus \{depot\}, t \in Taxi \qquad depart_v \leq t_{end} \quad (A.13)$$

$$\forall\, v \in V \qquad arrive_v \leq depart_v \quad (A.14)$$

Du temps passe à chaque traversée d'arc (A.15).

$$\forall u \in V, v \in V, t \in Taxi \begin{cases} arrive_v \leq depart_u + a(u,v).t + M \times (1 - x^t_{(u,v)}) \\ arrive_v \geq depart_u + a(u,v).t - M \times (1 - x^t_{(u,v)}) \end{cases} \quad (A.15)$$

Les fenêtres de temps des clients doivent être respectées (A.16) et être récupérés avant d'être déposés(A.17).

$$\forall\, r \in Request \begin{cases} r.\alpha \leq depart_{p_r} \leq r.\omega_1 \\ arrive_{d_r} \leq r.\omega_2 \end{cases} \quad (A.16)$$

$$\forall\, r \in Request \qquad depart_{p_r} \leq arrive_{d_r} \quad (A.17)$$

La recherche d'une place dure au maximum $searchmax$ (A.18). $\forall\, v \in V$, on définit une variable de décision supplémentaire $d_v \in \{0, 1\}$.

$$\forall\, v \in V, \begin{cases} search_v \leq searchmax \\ search_v \leq depart_v - arrive_v \\ search_v \geq searchmax - M \times d \\ search_v \geq depart_v - arrive_v - M \times (1 - d) \end{cases} \tag{A.18}$$

Ce système d'équations peut être remplacé par
$search_v == minl(searchmax, depart_v - arrive_v)$ en OPL.

A.3.4 Batterie

Les taxis commencent avec une batterie pleine (A.19) et ne peuvent pas avoir une batterie trop pleine (A.20).

$$\forall\, t \in Taxi, \qquad w_{beginning_t} = t.w \tag{A.19}$$

$$\forall\, t \in Taxi, u, v \in V \qquad w_v \leq t.w + M \times x^t_{(u,v)} \tag{A.20}$$

Les déplacements consomment de l'énergie et les charges en donnent (A.21).
$\forall\, u, v \in V, t \in Taxi$:

$$\begin{aligned} w_v \leq\ & w_u - t.multiplier \times (a_{(u,v)} + search_u) + M \times (1 - x^t_{(u,v)}) \\ & + (\text{ uniquement si } u = charge^i_j \in Charge) \\ & charger_j.w \times (depart_{charge^i_j} - search_{charge^i_j} - arrive_{charge^i_j}) \end{aligned} \tag{A.21}$$

Un chargeur ne peut pas être utilisé par plusieurs taxis simultanément.

$$\forall\, charge^i_j \in Charge \mid\ i < uses \qquad depart_{charge^i_j} \leq arrive_{charge^{i+1}_j} \tag{A.22}$$

Annexe B

Origine du recuit simulé

Voici l'exemple physique qui a inspiré le principe de recuit simulé. Les atomes forment naturellement des liaisons entre eux lorsqu'ils sont proches. Plus la température d'un atome est élevé, plus il bouge. Plus un atome bouge, moins les liaisons qu'il forme avec d'autres atomes tiennent. A faible température, un ensemble d'atomes forme un solide car les liaisons tiennent et l'ensemble adopte une structure rigide. A forte température, les liaisons ne tiennent plus, et l'ensemble devient liquide ou gazeux.

Certaines structures sont plus stables que d'autre. Avec un refroidissement instantané à un instant t d'un liquide, toute liaison formée après l'instant t est permanente. Un tel refroidissement génère des structures très instables et amorphes, contenant beaucoup de liaisons faibles (voir figure B.1). Cette structure peut être brisée facilement et redevenir liquide ou gazeuse à relativement faible température.

A l'inverse, avec un refroidissement progressif, les liaisons fortes deviennent rapidement dures à briser, tandis que les liaisons faibles restent faciles à briser pendant une période prolongée. Par sélection naturelle, seules les liaisons les plus fortes persistent, ce qui donne un solide cristallin, avec une structure régulière très stable (voir figure B.1). Le charbon a une structure amorphe, le diamant une structure cristalline.

Le principe d'un recuit en métallurgie consiste donc à réchauffer un solide, puis à le refroidir à une vitesse qui assure que le solide obtenu ait la bonne structure moléculaire. Dans un recuit simulé, l'objectif est de reproduire virtuellement ce phénomène physique, cette baisse progressive d'énergie du système afin d'obtenir la solution la plus stable possible. Commençant avec un système dans lequel tout changement est accepté, on va progressivement vers un système dans lequel aucun changement n'est accepté, en interdisant en priorité les changement fortement déstabilisants/dégradants. La figure B.2 présente un exemple d'utilisation d'un recuit simulé avec deux vitesses de refroidissement avec un refroidissement rapide et un lent. La solution obtenue par refroidissement lent est sensiblement meilleure.

Comme toute méta-heuristique, le recuit simulé mélange diversification et recherche locale, qu'on pourrait aussi appeler recherche macroscopique et microscopique, ou exploration et exploitation. La diversification consiste à étudier des solutions de structure très différentes, tandis que la recherche locale consiste à étudier des solutions ayant toutes peu ou prou la même forme. Dans le cas du recuit, il y a une transition progressive de la diversification vers la recherche locale. Plus le temps passe, plus la température diminue, et plus les modifications de solution sont restreintes.

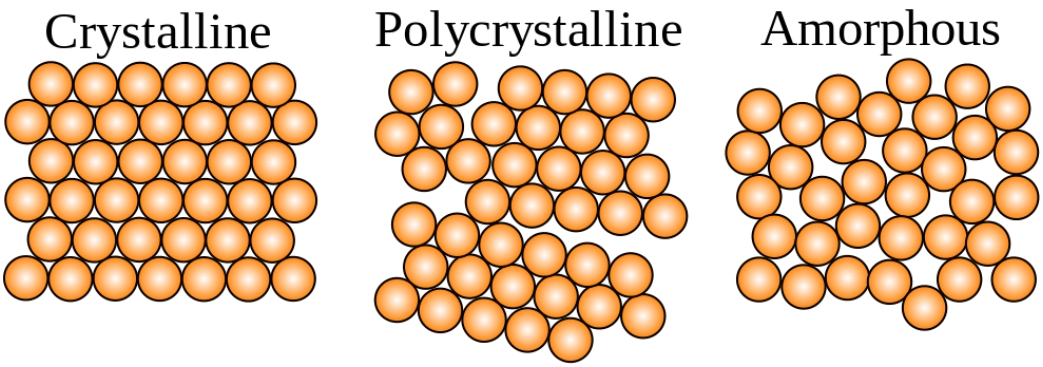


Figure B.1 – *Image extraite de Wikipedia. Trois types de structures moléculaires obtenues par solidification d'un liquide, avec différentes vitesse de refroidissement. Une structure cristalline très stable est obtenue par refroidissement lent, tandis qu'une structure amorphe est obtenue avec un refroidissement rapide.*

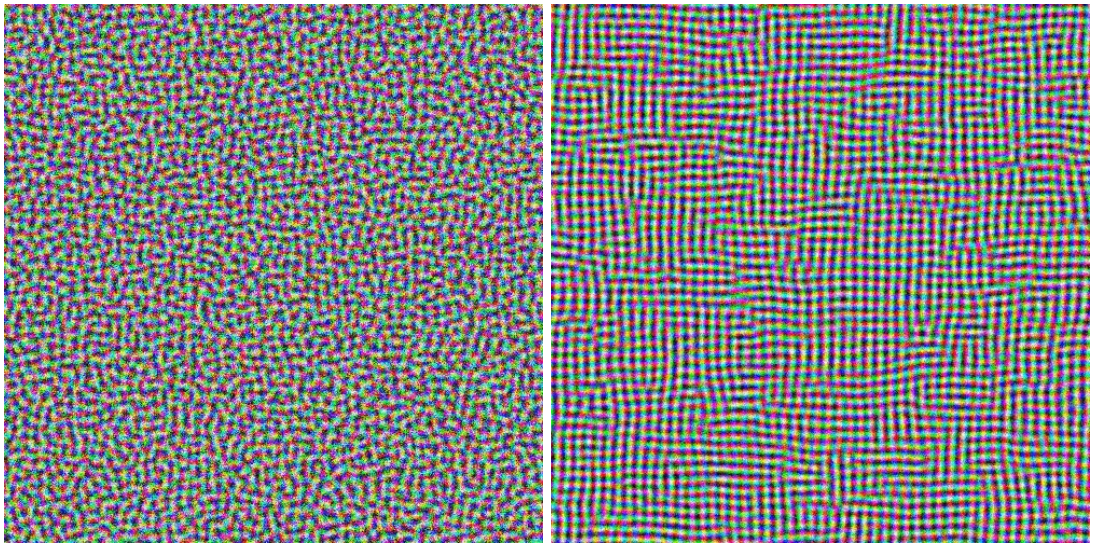


Figure B.2 – *Image extraite de Wikipedia. L'objectif est de positionner des billes de sorte qu'une bille ne soit pas adjacente à une bille de même couleur. Durant le recuit simulé, deux billes adjacentes peuvent échanger leur position. Ces échanges sont initialement libres, et plus le temps passe, plus les échanges diminuant la qualité de la solution fortement sont refusés. La figure de gauche (amorphe) est obtenue avec un refroidissement rapide, celle de droite (régulière) avec un refroidissement lent.*

Bibliographie

[Agatz et al., 2011] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2011). Dynamic ride-sharing : a simulation study in metro atlanta. *Transportation Research Part B-methodological - TRANSP RES PT B-METHOD*, 17.

[Agatz et al., 2012] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing : A review. In *European Journal of Operational Research*, volume 223, page 295–303.

[Akbari et al., 2003] Akbari, H., Shea Rose, L., and Taha, H. (2003). Analyzing the land cover of an urban environment using high-resolution orthophotos. *Landscape and Urban Planning*, 63(1) :1 – 14.

[Alex et al., 1989] Alex, K. P., Simon, K. A., and Samuel, K. A. (1989). *Simulated Annealing and Boltzmann Machines : A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley.

[Alonso-Mora et al., 2017] Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. In *Proceedings of the National Academy of Sciences*, volume 114, page 201611675.

[Alrefaei and Andradóttir, 1999] Alrefaei, M. H. and Andradóttir, S. (1999). A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management Science*, 45(5) :748–764.

[Andelmin and Bartolini, 2017] Andelmin, J. and Bartolini, E. (2017). An exact algorithm for the green vehicle routing problem. *Transportation Science*, 51(4) :1288–1303.

[Bai et al., 2021] Bai, R., Chen, X., Chen, Z.-L., Cui, T., Gong, S., He, W., Jiang, X., Jin, H., Jin, J., Kendall, G., Li, J., Lu, Z., Ren, J., Weng, P., Xue, N., and Zhang, H. (2021). Analytics and machine learning in vehicle routing research.

[Bauer et al., 2018] Bauer, G., B. Greenblatt, J., and F. Gerke, B. (2018). Cost, energy, and environmental impact of automated electric taxi fleets in manhattan. In *Environmental Science & Technology*, volume 52.

[Behnke et al., 2021] Behnke, M., Kirschstein, T., and Bierwirth, C. (2021). A column generation approach for an emission-oriented vehicle routing problem on a multigraph. *European Journal of Operational Research*, 288(3) :794–809.

[Belloche, 2015] Belloche, S. (2015). On-street parking search time modelling and validation with survey-based data. *Transportation Research Procedia*, 6 :313 – 324. 4th International Symposium of Transport Simulation (ISTS'14) Selected Proceedings, Ajaccio, France, 1-4 June 2014.

[Ben Ticha et al., 2018] Ben Ticha, H., Absi, N., Feillet, D., Quilliot, A., and Van Woensel, T. (2018). A branch-and-price algorithm for the vehicle routing problem with time windows on a road network. *Networks*, 73.

[Berbeglia et al., 2007] Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems : A classification scheme and survey. In *TOP : An Official Journal of the Spanish Society of Statistics and Operations Research*, volume 15, pages 1–31.

[Berbeglia et al., 2010] Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2010). Dynamic pickup and delivery problems. In *European Journal of Operational Research*, volume 202, pages 8–15.

[Bertsimas et al., 2019] Bertsimas, D., Jaillet, P., and Martin, S. (2019). Online vehicle routing : The edge of optimization in large-scale applications. *Operations Research*, pages 143–162.

[Billhardt et al., 2017] Billhardt, H., Fernández, A., Lujak, M., Ossowski, S., Julián, V., De Paz, J., and Hernández, J. (2017). Coordinating open fleets. a taxi assignment example. In *AI Communications*, volume 30, pages 1–16.

[Bongiovanni et al., 2018] Bongiovanni, C., Kaspi, M., and Geroliminis, N. (2018). A two phase heuristic approach for the dynamic electric autonomous dial-a-ride problem. *7th Symposium of the European Association for Research in Transportation (hEART)*.

[Bongiovanni et al., 2019] Bongiovanni, C., Kaspi, M., and Geroliminis, N. (2019). The electric autonomous dial-a-ride problem. *Transportation Research Part B : Methodological*, 122 :436–456.

[Braekers et al., 2016] Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem : State of the art classification and review. *Computers & Industrial Engineering*, 99 :300 – 313.

[Burghout et al., 2015] Burghout, W., Rigole, P.-J., and Andréasson, I. (2015). Impacts of shared autonomous taxis in a metropolitan area. *Proceedings of the 94th annual meeting of the Transportation Research Board*.

[Bösch et al., 2018] Bösch, P. M., Becker, F., Becker, H., and Axhausen, K. W. (2018). Cost-based analysis of autonomous mobility services. *Transport Policy*, 64 :76 – 91.

[Caicedo et al., 2012] Caicedo, F., Blazquez, C., and Miranda, P. (2012). Prediction of parking space availability in real time. *Expert Systems with Applications*, 39(8) :7281 – 7290.

[Cao et al., 2018] Cao, Y., Zhang, X., Liu, W., Cao, Y., Chiaraviglio, L., Wu, J., and Putrus, G. (2018). Reservation based electric vehicle charging using battery switch. *Proceedings of the IEEE International Conference on Communications (ICC)*.

[Chester et al., 2010] Chester, M., Horvath, A., and Madanat, S. (2010). Parking infrastructure : energy, emissions, and automobile life-cycle environmental accounting. *Environmental Research Letters*, 5(3) :034001.

[Clarke and Wright, 1964] Clarke, G. and Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12 :568–581.

[Cohn and Fielding, 1999] Cohn, H. and Fielding, M. (1999). Simulated annealing : Searching for an optimal temperature schedule. *SIAM Journal on Optimization*, 9(3) :779–802.

[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 (darp) : Models and algorithms. In *Annals OR*, volume 153, pages 29–46.

[Croes, 1958] Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6) :791–812.

[Cui et al., 2019] Cui, Q., Weng, Y., and Tan, C. (2019). Electric vehicle charging station placement method for urban areas. *IEEE Transactions on Smart Grid*, 10(6) :6552–6565.

[Darbéra, 2010] Darbéra, R. (2010). Taxicab regulation and urban residents' use and perception of taxi services : a survey in eight cities. In *Proceedings of the 12th WCTR, July 11-15, 2010 – Lisbon, Portugal*.

[David et al., 2006] David, A., Robert, B., Vasek, C., and William, C. (2006). *The Travelling Salesman Problem*. Princeton University.

[Davis et al., 2010] Davis, A. Y., Pijanowski, B. C., Robinson, K., and Engel, B. (2010). The environmental and economic costs of sprawling parking lots in the united states. *Land Use Policy*, 27(2) :255 – 261. Forest transitions Wind power planning, landscapes and publics.

[Delos Reyes et al., 2016] Delos Reyes, J. R. M., Parsons, R. V., and Hoemsen, R. (2016). Winter happens : The effect of ambient temperature on the travel range of electric vehicles. *IEEE Transactions on Vehicular Technology*, 65(6) :4016–4022.

[Delucchi, 1997] Delucchi, M. A. (1997). The annualized social cost of motor vehicle use in the u.s.-based on 1990–1991 data : Summary of theory, data, methods, and results. In Hohmeyer, O., Rennings, K., and Ottinger, R. L., editors, *Social Costs and Sustainability*, pages 380–417, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Desaulniers et al., 2016] Desaulniers, G., Errico, F., Irnich, S., and Schneider, M. (2016). Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research*, 64.

[Dey and Molinaro, 2018] Dey, S. S. and Molinaro, M. (2018). Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170 :237–266.

[Dueck and Scheuer, 1990] Dueck, G. and Scheuer, T. (1990). Threshold accepting : A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1) :161–175.

[Eksioglu et al., 2009] Eksioglu, B., Vural, A. V., and Reisman, A. (2009). The vehicle routing problem : A taxonomic review. *Computers & Industrial Engineering*, 57(4) :1472 – 1483.

[Erdoğan and Miller-Hooks, 2012] Erdoğan, S. and Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation Research Part E : Logistics and Transportation Review*, 109 :100–114.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*.

[Felipe et al., 2014] Felipe, A., Ortuño, M. T., Righini, G., and Tirado, G. (2014). A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research Part E : Logistics and Transportation Review*, 71 :111 – 128.

[Fielding, 2000] Fielding, M. (2000). Simulated annealing with an optimal fixed temperature. *SIAM Journal on Optimization*, 11(2) :289–307.

[Fraile-Ardanuy et al., 2018] Fraile-Ardanuy, J., Castano-Solis, S., Álvaro Hermana, R., Merino, J., and Ángela Castillo (2018). Using mobility information to perform a feasibility study and the evaluation of spatio-temporal energy demanded by an electric taxi fleet. *Energy Conversion and Management*, 157 :59 – 70.

[Franz et al., 2001] Franz, A., Hoffmann, K. H., and Salamon, P. (2001). Best possible strategy for finding ground states. *Phys. Rev. Lett.*, 86 :5219–5222.

[Fukasawa et al., 2006] Fukasawa, R., Longo, H., Lysgaard, J., Poggi, M., Reis, M., Uchoa, E., and Werneck, R. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Program.*, 106 :491–511.

[Gacias and Meunier, 2012] Gacias, B. and Meunier, F. (2012). Operating a fleet of electric taxis.

[Garey et al., 1979] Garey, M., Johnson, D., and Collection, M. S. M. (1979). *Computers and Intractability : A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman.

[Gearhart et al., 2013] Gearhart, J. L., Adair, K. L., Durfee, J. D., Jones, K. A., Martin, N., and Detry, R. J. (2013). Comparison of open-source linear programming solvers. *Web*.

[Goeke and Schneider, 2015] Goeke, D. and Schneider, M. (2015). Routing a mixed fleet of electric and conventional vehicles. *European Journal of Operational Research*, 245(1) :81 – 99.

[Granville et al., 1994] Granville, V., Krivanek, M., and Rasson, J. . (1994). Simulated annealing : a proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6) :652–656.

[Guillou et al., 2011] Guillou, H., Cung, V.-D., Ha, D. L., Jacomino, M., and Merten, J. (2011). Energy management strategies for optimal charging of electric vehicles with photovoltaic production. In *The 26th European photovoltaic solar energy conference and exhibition*, pages 3885 – 3889, Hambourg, Germany.

[Gurumurthy and Kockelman, 2018] Gurumurthy, K. M. and Kockelman, K. M. (2018). Analyzing the dynamic ride-sharing potential for shared autonomous vehicle fleets using cellphone data from orlando, florida. *Computers, Environment and Urban Systems*, 71 :177 – 185.

[Ha et al., 2013] Ha, D. L., Guillou, H., Cung, V.-D., and Jacomino, M. (2013). Patent wo2013014238.

[Ha et al., 2014] Ha, D. L., Guillou, H., Cung, V.-D., and Jacomino, M. (2014). Patent wo2014037356.

[Hastings, 1970] Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1) :97–109.

[Hiermann et al., 2016] Hiermann, G., Puchinger, J., Ropke, S., and Hartl, R. F. (2016). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, 252(3) :995 – 1018.

[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.

[Hoché et al., 2020] Hoché, T., Barth, D., Mautor, T., and Burghout, W. (2020). Charging management of shared taxis : Neighbourhood search for the e-adarp. *IEEE International Conference on Intelligent Transportation Systems*.

[Houissa et al., 2017] Houissa, A., Barth, D., Faul, N., and Mautor, T. (2017). A learning algorithm to minimize the expectation time of finding a parking place in urban area. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 29–34.

[Hu et al., 2018] Hu, L., Dong, J., Lin, Z., and Yang, J. (2018). Analyzing battery electric vehicle feasibility from taxi travel patterns : The case study of new york city, usa. In *Transportation Research Part C : Emerging Technologies*, volume 87, pages 91–104.

[Jorge and Homem de Almeida Correia, 2013] Jorge, D. and Homem de Almeida Correia, G. (2013). Carsharing systems demand estimation and defined operations : A literature review. In *European Journal of Transport and Infrastructure Research*, volume 13, pages 201–220.

[Kallehauge, 2006] Kallehauge, B. (2006). On the vehicle routing problem with time windows. *Column generation*, pages 67–98.

[Karpinski et al., 2015] Karpinski, M., Lampis, M., and Schmied, R. (2015). New inapproximability bounds for tsp. *Journal of Computer and System Sciences*, 81(8) :1665–1677.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Jr., C. D. G., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 :671–680.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1□2) :83–97.

[Lam et al., 2014] Lam, A. Y. S., Leung, Y., and Chu, X. (2014). Electric vehicle charging station placement : Formulation, complexity, and solutions. *IEEE Transactions on Smart Grid*, 5(6) :2846–2856.

[Lawler et al., 1991] Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1991). *The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization*. Wiley.

[Lefauconnier and Gantelet, 2005] Lefauconnier, A. and Gantelet, E. (2005). La recherche d'une place de stationnement : Strategies, nuisances associees, enjeux pour la gestion du stationnement en france. In *TEC (TRANSPORT, ENVIRONNEMENT, CIRCULATION)*, volume 187, pages 9–12.

[Li et al., 2016] Li, Z., Jiang, S., Dong, J., Wang, S., Ming, Z., and Li, L. (2016). Battery capacity design for electric vehicles considering the diversity of daily vehicles miles traveled. *Transportation Research Part C : Emerging Technologies*, 72 :272 – 282.

[Liang et al., 2016] Liang, X., de Almeida Correia, G. H., and van Arem, B. (2016). Optimizing the service area and trip selection of an electric automated taxi system used for the last

mile of train trips. *Transportation Research Part E : Logistics and Transportation Review*, 93 :115 – 129.

[Liao et al., 2016] Liao, B., Li, L., Li, B., Mao, J., Yang, J., Wen, F., and Salam, M. (2016). Load modeling for electric taxi battery charging and swapping stations : Comparison studies. In *IEEE 2nd Annual Southern Power Electronics Conference (SPEC)*, pages 1–6.

[Lin et al., 2016] Lin, J., Zhou, W., and Wolfson, O. (2016). Electric vehicle routing problem. *Transportation Research Procedia*, 12 :508 – 521. Tenth International Conference on City Logistics 17-19 June 2015, Tenerife, Spain.

[Lioris and Cohen, 2016] Lioris, J. E. and Cohen, G. (2016). Optimised flexible transport mode without advanced reservations by a des approach. *IFAC-PapersOnLine*, 49(18) :974 – 979. 10th IFAC Symposium on Nonlinear Control Systems NOLCOS 2016.

[Locatelli, 2000] Locatelli, M. (2000). Simulated annealing algorithms for continuous global optimization : Convergence conditions. *Journal of Optimization Theory and Applications*, 104 :121–133.

[M. Sweda et al., 2017] M. Sweda, T., S. Dolinskaya, I., and Klabjan, D. (2017). Adaptive routing and recharging policies for electric vehicles. *Transportation Science*, 51.

[Marinakis et al., 2018] Marinakis, Y., Marnakis, M., and Migdalas, A. (2018). Particle swarm optimization for the vehicle routing problem : A survey and a comparative analysis. *Handbook of Heuristics*, pages 1163–1196.

[Martinez et al., 2014] Martinez, L., Homem de Almeida Correia, G., and Viegas, J. (2014). An agent-based simulation model to assess the impacts of introducing a shared-taxi system : An application to lisbon (portugal). In *Journal of Advanced Transportation*, volume 49.

[Merve and Bülent, 2016] Merve, K. and Bülent, C. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research Part C : Emerging Technologies*, 65 :111 – 127.

[Mohamed and Thomas, 2014] Mohamed, S. H. and Thomas, S. (2014). Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Computers and Operations Research*, 43 :286 – 291.

[Mommens et al., 2018] Mommens, K., Lebeau, P., Verlinde, S., van Lier, T., and Macharis, C. (2018). Evaluating the impact of off-hour deliveries : An application of the transport agent-based model. *Transportation Research Part D : Transport and Environment*, 62 :102 – 111.

[Montoya et al., 2017] Montoya, A., Guéret, C., Mendoza, J. E., and Villegas, J. G. (2017). The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B : Methodological*, 103 :87 – 110. Green Urban Transportation.

[Morganti et al., 2014] Morganti, E., Dablanc, L., and Fortin, F. (2014). Final deliveries for online shopping : The deployment of pickup point networks in urban and suburban areas. *Research in Transportation Business & Management*, 11 :23 – 31. Managing Freight in Urban Areas.

[Muelas et al., 2015] Muelas, S., LaTorre, A., and Peña, 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.

[Nikolaev and Jacobson, 2010] Nikolaev, A. and Jacobson, S. (2010). *Simulated Annealing*, volume 146, pages 1–39. Springer.

[Ntafos and Gonzalez, 1984] Ntafos, S. and Gonzalez, T. (1984). On the computational complexity of path cover problems. *Journal of Computer and System Sciences*, 29(2) :225 – 242.

[Papadimitriou, 1992] Papadimitriou, C. H. (1992). The complexity of the lin–kernighan heuristic for the traveling salesman problem. *SIAM Journal on Computing*, 21(3) :450–465.

[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 and Schmid, 2013] Parragh, S. N. and Schmid, V. (2013). Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1) :490–497.

[Pelletier et al., 2018] Pelletier, S., Jabali, O., and Laporte, G. (2018). Charge scheduling for electric freight vehicles. *Transportation Research Part B : Methodological*, 115 :246 – 269.

[Pillac et al., 2013] Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1) :1 – 11.

[Pisinger and Ropke, 2005] Pisinger, D. and Ropke, S. (2005). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34 :2403–2435.

[Polak and Axhausen, 1990] Polak, J. and Axhausen, K. (1990). Parking search behaviour : A review of current research and future prospects.

[Potra and Wright, 2000] Potra, F. A. and Wright, S. J. (2000). Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1) :281–302. Numerical Analysis 2000. Vol. IV : Optimization and Nonlinear Equations.

[Preeti and Prasenjit, 2018] Preeti, W. and Prasenjit, S. (2018). *Car Sharing Market Size By Model (P2P, Station-Based, Free-Floating), By Business Model (Round Trip, One Way), By Application (Business, Private), Industry Analysis Report, Regional Outlook, Growth Potential, Competitive Market Share & Forecast, 2018 – 2024*. Global Market Insights.

[Raman and Gill, 2017] Raman, V. and Gill, N. S. (2017). Review of different heuristic algorithms for solving travelling salesman problem. *International Journal of Advanced Research in Computer Science*, 8(5) :423–425.

[Robini and Reissman, 2013] Robini, M. and Reissman, P. (2013). From simulated annealing to stochastic continuation : a new trend in combinatorial optimization. *Journal of Global Optimization*, 56 :185–215.

[Ronald et al., 2015] Ronald, N., Thompson, R., and Winter, S. (2015). Simulating demand-responsive transportation : A review of agent-based approaches. *Transport Reviews*, 35(4) :404–421.

[Savelsbergh and Sol, 1995] Savelsbergh, M. and Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, 29 :17–29.

[Schneider et al., 2014] Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, pages 500–520.

[Shoup, 2005] Shoup, D. (2005). The high cost of free parking. *Journal of Planning Education and Research*, 17 :3–20.

[Simchi-Levi et al., 2014] Simchi-Levi, D., Chen, X., and Bramel, J. (2014). *Solving the VRP Using a Column-Generation Approach*, pages 359–375. Springer New York, New York, NY.

[Skiena, 2008] Skiena, S. S. (2008). *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition.

[Solomon, 1987] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2) :254–265.

[Stone and Tovey, 1991] Stone, R. E. and Tovey, C. A. (1991). The simplex and projective scaling algorithms as iteratively reweighted least squares methods. *SIAM Review*, 33(2) :220–237.

[Taefi, 2016] Taefi, T. (2016). Viability of electric vehicles in combined day and night delivery : A total cost of ownership example in germany. In *European Journal of Transportation and Infrastructure Research*, volume 16, pages 512–553.

[Tsao et al., 2018] Tsao, M., Iglesias, R., and Pavone, M. (2018). Stochastic model predictive control for autonomous mobility on demand. unpublished.

[Ukkusuri et al., 2015] Ukkusuri, S., Ozbay, K., Yushimito, W., Morgul, E., iyer, S., and Holguín-Veras, J. (2015). Assessing the impact of urban off-hour delivery program using city scale simulation models. In *EURO Journal on Transportation and Logistics*, volume 5, pages 1–26.

[website, 1] website (1). https://www.statistiques.developpement-durable.gouv.fr/sites/default/files/2018-10/datalab-31-chiffres-cles-transport-mars2018-c.pdf.

[website, 10] website (10). https://www.openstreetmap.org.

[website, 11] website (11). https ://sumo.dlr.de.

[website, 12] website (12). http://www.vissim.com.

[website, 13] website (13). https://en.wikipedia.org/wiki/Annealing_(metallurgy).

[website, 14] website (14). https://blogs.worldbank.org/transport/5g-and-transport-envisioning-possibilities-better-connected-tomorrow.

[website, 2] website (2). https://en.wikipedia.org/w/index.php?title=Fossil_fuel_phase-out.

[website, 5] website (5). https://www.mordorintelligence.com/industry-reports/ridesharing-market.

[website, 7] website (7). http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/.

[website, 8] website (8). https://www.insee.fr/fr/statistiques/3676874?sommaire=3696937.

[website, 9] website (9). http://www.senat.fr/rap/r20-313/r20-3130.html.

[zzz15, 15] zzz15 (15). https://en.wikipedia.org/wiki/Inductive_charging.