Optimisation et intégration de la mobilité partagée dans les systèmes de transport multimodaux

Kamel Aissat

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Optimisation et intégration de la mobilité partagée dans les systèmes de transport multimodaux

THÈSE

présentée et soutenue publiquement le 04 Avril 2016

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Kamel AISSAT

Composition du jury

Rapporteurs :	Roberto Wolfler Calvo	Professeur, Université de Paris Nord
	Marie-José Huguet	Maître de conférence (HDR), Institut National des Sciences Appliquées de Toulouse
Examinateurs :	Ammar Oulamara	Professeur, Université de Lorraine (directeur de thèse)
	Sacha Varone	Professeur, University of Applied Sciences and Arts Western Switzerland
	Olivier Péton	Maître Assistant (HDR), École des Mines de Nantes
	Zineb Habbas	Professeur, Université de Lorraine
Invité :	Marc Grojean	Directeur de l'entreprise Covivo

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

C'est avec une certaine émotion et beaucoup de sincérité que je voudrais remercier toutes les personnes ayant soutenu et apprécié mon travail.
En premier lieu, je tiens à exprimer mes plus vifs remerciements à Monsieur Ammar Oulamara qui fut pour moi un directeur de thèse attentif et disponible malgré ses nombreuses charges. Sa compétence, sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris.

J'exprime tous mes remerciements à l'ensemble des membres de mon jury : Mesdames Huguet Marie-José, Zineb Habbas et Messieurs Roberto Wolfler Calvo, Sacha Varone, Olivier Péton et Marc Grojean.
Un grand merci à Monsieur Sacha Varone pour notre collaboration qui a été très enrichissante pour moi.
Je remercie toutes les personnes formidables que j'ai rencontrées par le biais de ma thèse. Je pense tout particulièrement aux membres et ex-membres de l'équipe de COVIVO et ORCHIDS pour le climat sympathique dans lequel ils m'ont permis de travailler. Leur gentillesse, leur compétence et leur humour, les nombreuses discussions que j'ai pu avoir avec chacun m'ont beaucoup apporté. Une grande pensée à ma femme qui m'a tant aidé, supporté et permis de me lever motivé, le coeur léger et l'esprit tranquille depuis le début de ma thèse. Je ne saurais terminer sans remercier toutes ces personnes dans l'ombre dont la contribution à mon travail est non négligeable.
Enfin, les mots les plus simples étant les plus forts, j'adresse toute mon affection à ma famille, et en particulier à mes parents et à mon frère. Malgré mon éloignement, leur intelligence, leur confiance, leur tendresse leur amour me portent et me guident tous les jours.

Sommaire

.

Table des figures

1

Introduction générale

1.1 Introduction

Plusieurs villes à travers le monde sont confrontées à des problèmes de mobilité ce qui implique des problèmes de congestion, d'augmentation de la consommation d'énergie et des émissions de gaz à effet de serre. Ces effets ont un coût important sur l'économie mais aussi des conséquences néfastes sur l'environnement et sur la qualité de vie des citoyens.

Une analyse approfondie des données statistiques d'émissions de gaz à effet de serre permet d'observer que l'activité des transports est responsable de la majorité des émissions de gaz en France. Plus précisément, l'émission de carbone due aux activités de transport représente 34,5% des émissions totales, loin devant les émissions dues aux résidences et bâtiments avec 21,1%, et les émissions de l'industrie avec 19,4% [1]. En ce qui concerne l'activité des transports, 51.2% des émissions sont dues aux véhicules des particuliers [2]. Ce pourcentage n'est pas surprenant quand on sait qu'en 2010, 81.8% des passagers-kilomètres sont effectués par des véhicules privés, alors que seulement 5,6% sont réalisés par des bus et 11,2% par rail [3]. En outre, la plupart des déplacements en véhicule privé ont été réalisés avec un taux d'occupation très faible, à savoir : une personne par véhicule. Ainsi, le coût des sièges vides est extrêmement élevé. En se basant sur les hypothèses mentionnées dans le rapport du développement durable [4], le taux d'occupation moyen est de 1.3 personnes par voiture et de 0,05 euro/kilomètre. Le coût des déplacements à vide représente environ 39.5 milliards d'euros par an en France. Ainsi, il existe une marge importante pour la réduction des coûts grâce au développement d'initiatives qui incite à une meilleure occupation des véhicules particuliers. Il en résulte de nouveaux défis et initiatives pour :
(i) la réduction de l'importance de voitures privées dans la chaîne de mobilité.
(ii) l'utilisation intelligente des voitures particulières dans lesquelles une voiture sera considérée comme un moyen de mobilité.

Le besoin de ces initiatives se confirme à travers les enquêtes d'opinion où l'usager est déjà conscient et même prêt à les adopter. Par exemple, le rapport d'étude de l'observatoire Cetelem de l'automobile [5] indique que le véhicule automobile est en train de devenir le transport en com-

1. Statistiques de 2008 service de l'observation et des statistiques (Soes)-Commissariat Général au développement
2. Statistiques de l'Observation et des statistiques (Soes)-Commissariat Général au développement Durable. Plus d'informations disponible sur http ://www.developpement-durable.gouv.fr/Commissariat-general-au.html
3. http ://www.developpement-durable.gouv.fr/Commissariat-general-au.html
4. http ://www.developpement-durable.gouv.fr/Commissariat-general-au.html
5. Une enquête réalisée par TNS Sofres, 4 830 personnes interrogées par Internet en septembre 2013 dans huit pays

mun de l'avenir. Ce dernier reste incontournable dans la mobilité quotidienne des Européens dont 74% d'entre eux pensent que dans dix ans il conservera une importance supérieure ou égale à celle d'aujourd'hui. D'après cette enquête, près d'un citoyen sur deux semble imaginer qu'à long terme, la voiture deviendra un bien partagé « 73% des Européens voient le covoiturage et l'autopartage se développer au cours des prochaines années ».

Dans cette perspective de nouvelles mobilités, plusieurs initiatives en complément des transports en commun traditionnels, ont vu le jour ces dernières années. Parmi celles-ci, on trouve le covoiturage comme l'atteste la réussite des start-up Blablacar [6] et Uber [7], mais aussi des initiatives de transports en commun multimodaux (incluant plusieurs modes de déplacement comme le bus, le tramway, le métro, la marche à pied et le vélo). Ces succès ont été assurés par la démocratisation des outils de "Technologies de l'Information et de la Communication" (TIC), notamment à travers les smartphones et les fonctionnalités de géolocalisation. Ces derniers ont largement accéléré l'émergence de nouveaux services de transport, comme par exemple le service d'Uber-Van dans le monde, le service Padambus [8] pour les noctambules à Paris, et le service Tooxme [9] en Suisse.

Les deux initiatives (covoiturage et transport en commun multimodaux) sont développées séparément, et il n'existe pas actuellement d'offres de mobilité incluant le covoiturage associée à un transport en commun. En effet, dans une étude exploratoire "MITRA" [10] (Multimodality and Interoperability for sustainable TRAnsport in commuting situation), les auteurs concluent que le concept d'un Système d'Information Multimodal (SIM) mobile instantané combinant à la fois le covoiturage et le transport en commun multimodal est une véritable innovation et qu'il n'y a pas de service/application comparable. Ils présentent également plusieurs défis pour la conception et la mise en œuvre d'un tel service. Parmi ces défis, un tel SIM devrait intégrer des composants de calculs d'itinéraires incluant le covoiturage dynamique et le transport en commun tout en offrant une qualité de service élevée (temps de réponse, nombre d'itinéraires alternatifs, etc). Le composant de calcul d'itinéraire sera le noyau du système d'information multimodal. Ce dernier est relativement indépendant des autres composants d'un service SIM qui doit inclure un appariement dynamique entre les conducteurs et les passagers.

1.2 Objectifs de la thèse et organisation du manuscrit

A ce jour, les différents modes de transport existant sont sous-exploités et ne répondent pas en totalité aux besoins des utilisateurs ainsi qu'aux enjeux environnementaux.
Dans cette thèse, nous développons différentes techniques d'optimisation qui permettent de renforcer la mobilité individuelle et partagée en répondant à la fois aux exigences des utilisateurs et de l'environnement. Autrement dit, nous nous positionnons dans un contexte favorable aux modes de déplacement alternatifs aux transport publics. L'originalité de cette thèse est le développement de briques élémentaires pour un service de mobilité unifié intégrant la mobilité partagée et les trans-

d'Europe (Allemagne, France, Italie, Portugal, Espagne, Belgique, Royaume-Unis et Turquie) avec des échantillons représentatifs des populations nationales d'au moins 600 personnes par pays. Les analyses et les prévisions ont été réalisées en novembre 2013 en partenariat avec le cabinet d'étude et de conseil BIPE. L'étude complète disponible sur http ://observatoirecetelem.com/lobservatoire-cetelem/

6. www.blablacar.fr
7. https ://www.uber.com/fr/
8. www.padambus.com
9. http ://tooxme.com/
10. Étude présentée en 2010, www.ratp.fr

ports en commun en temps réel. La suite de ce mémoire est organisée comme suit :

Dans le chapitre 2, nous décrivons dans un premier temps l'intérêt de la théorie des graphes dans les problèmes de cheminement, en particulier le problème du plus court chemin. Pour cela, nous commençons par des généralités autour des graphes suivi d'une présentation de quelques algorithmes de plus court chemin ainsi que des techniques d'accélération. Ensuite, nous présentons les travaux existants dans la littérature sur les différents types de mobilité partagée ainsi que les problèmes connexes (en détaillant les différentes variantes du service de covoiturage).

Dans le chapitre 3, nous nous intéressons au problème de covoiturage avec des emplacements intermédiaires de prise en charge et de dépose pour les passagers. Ce dernier est une généralisation du problème de covoiturage où les emplacements de prise en charge et de dépose peuvent différer de l'origine et de la destination des passagers. Dans ce premier modèle, les modes de transport d'un passager pour se rendre à l'emplacement de prise en charge depuis son origine ou à partir de l'emplacement de dépose vers sa destination, ne doivent pas dépendre du temps. Ces derniers correspondent généralement aux véhicules privés, marche à pied, vélos, etc.
Ce travail a donné lieu à deux communications en conférence internationale [Aissat and Oulamara, 2014a] et [Aissat and Oulamara, 2015a] et à un journal [Aissat and Oulamara, 2014b].

Dans le chapitre 4, nous présentons le problème de covoiturage aller-retour avec stations relais. Contrairement au covoiturage classique aller-retour, ce dernier permet de prendre en considération d'éventuelles stations relais où le passager peut laisser son véhicule privé sur le trajet aller en lui garantissant un trajet retour par un autre covoiturage via cette station relais afin de récupérer son propre véhicule.
Ce travail a donné lieu à une communication en conférence internationale [Aissat and Oulamara, 2015b].

Le chapitre 5 est dédié au problème dit Dynamic Dial-a-Ride appliqué à une centrale de mobilité. Dans ce modèle, nous nous positionnons dans le cadre d'une centrale de taxis où les requêtes des passagers arrivent en temps réel et doivent s'insérer dans les trajets des taxis.
Nous étendons ensuite ce modèle pour prendre en compte une flotte mixte composée de taxis-partagés et de covoiturage.

Dans le chapitre 6, nous développons une approche qui permet d'allier le transport privé au transport public en se basant sur un itinéraire multimodal classique. Plus exactement, il combine le covoiturage avec les transports en commun en considérant uniquement les arrêts de correspondances comme lieux potentiels de prise en charge et de dépose des passagers. Contrairement au modèle décrit dans le quatrième chapitre, les modes de transport d'un usager ne sont pas limités à ceux qui ne dépendent pas du temps (à savoir, la prise en compte des transports publics, etc).
Ce travail a donné lieu à deux communications en conférence internationale [Varone and Aissat, 2015] [Aissat and Varone, 2015b] et un chapitre d'ouvrage [Aissat and Varone, 2015a].

Enfin dans le chapitre 7, nous présentons nos conclusions ainsi que quelques variantes aux problèmes que nous avons traités.

2

Revue de la littérature

Sommaire

Dans ce chapitre, nous nous intéressons au contexte scientifique et à la modélisation des problèmes de cheminement. Dans un premier temps, nous présentons quelques définitions et propriétés sur les graphes. Dans un deuxième temps, nous introduisons le problème de plus court chemin dans des graphes statiques, en nous focalisant sur les deux algorithmes les plus utilisés, à savoir l'algorithme de Dijkstra [Dijkstra, 1959] et l'algorithme de Bellman-Ford [Bellman, 1956]. En partant de ces algorithmes, nous verrons ensuite les différentes techniques d'accélération utilisées dans les graphes statiques. Ensuite, nous présentons quelques types de modélisation pour les graphes non statiques (dynamiques). Enfin, nous présentons une étude bibliographique sur la mobilité partagée.

2.1 Modélisation par la théorie des graphes

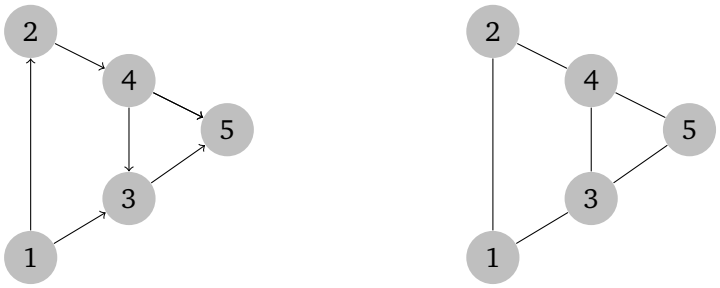
2.1.1 Graphes

Les graphes sont des concepts mathématiques utilisés pour modéliser les relations entre des objets d'un même ensemble. Ils sont fréquemment utilisés pour modéliser des systèmes qui se présentent sous la forme d'un réseau tels que les télécommunications, la planification, les transports ou encore la théorie de la complexité. Autrement dit, un graphe définit l'existence d'une relation entre les objets tels qu'une ligne entre deux stations de métro, une relation d'amitié dans un réseau social ou encore une rue entre deux carrefours. Il existe deux types de graphes : les graphes orientés et les graphes non orientés.

Definition. 1 *Un graphe G est un couple (N,A) avec N un ensemble dont les éléments sont appelés nœuds (sommets) et A un ensemble dont les éléments sont des paires de nœuds (sommets). Si A correspond*

à un ensemble d'arêtes, le graphe est dit non-orienté. En revanche, si A correspond à un ensemble d'arcs, le graphe est dit orienté. Une arête (i, j) *signifie qu'il est possible de passer du nœud i vers le nœud j et également du nœud j vers le nœud i, tandis qu'un arc* (i, j) *signifie qu'il est possible de passer uniquement du nœud i vers le nœud j. Les cardinaux associés aux deux ensembles N et A sont respectivement n et m. Ces derniers représentent respectivement le nombre de nœuds (sommets) et d'arcs (arêtes) du graphe G.*

La figure 2.1 présente un exemple d'un graphe orienté et non orienté.



(a) Graphe orienté (b) Graphe non orienté

FIGURE 2.1 – Exemple d'un graphe orienté et non orienté.

Successeurs et prédécesseurs
La fonction Γ^+ définit pour chaque noeud les successeurs, soit $\Gamma^+(u) = \{v|(u,v) \in A\}$. Symétriquement on définit les prédécesseurs $\Gamma^-(u) = \{v|(v,u) \in A\}$. Lorsque le graphe est symétrique, les prédécesseurs et successeurs d'un nœud sont les mêmes. Plus généralement, deux nœuds sont dit adjacents lorsqu'ils sont reliés par un arc ou une arête.

Graphe pondéré
Un graphe *G* est dit pondéré si une fonction de poids lui est associée. Cette fonction peut prendre différentes formes, et suivant sa nature, plusieurs classes de graphes peuvent être définies. De manière générale, le poids d'un arc appartient à l'ensemble des nombres réels. Dans ce qui suit, la fonction de coût sera supposée positive. Cette hypothèse a été considérée car elle permet de refléter la réalité des problèmes traités dans notre thèse.

Degré et densité
Le degré d'un nœud est le nombre d'arcs adjacents à ce nœud, et la densité d'un graphe est définie par le rapport entre le nombre d'arêtes (ou d'arcs) divisé par le nombre d'arêtes (ou d'arcs) possibles, soit $\frac{2|A|}{|N|\cdot|N-1|}$. Nous pouvons prendre comme exemple le réseau routier correspondant à l'ile de Manhattan (figure 2.2) où les rues se croisent toujours en un carrefour à quatre branches et dont les rues sont généralement à double sens. En utilisant un graphe orienté, le degré moyen d'un nœud est proche de 8, soit deux arcs par rue du carrefour. Dans ce même cas, la densité du graphe sera de l'ordre de 4.

FIGURE 2.2 – Exemple de modélisation d'un réseau routier sous forme d'un graphe (une partie de l'île de Manhattan).

Notions de chemin, chaîne, cycle et circuit

Dans un graphe il est naturel de vouloir se déplacer de nœud en nœud en suivant les arcs ou les arêtes du graphe. Dans le cas où le déplacement s'effectue sur un ensemble d'arcs, une telle marche est appelée un *chemin*. Dans le cas des arêtes, la marche correspondante est une *chaîne*.

Definition. 2 *Un chemin (ou itinéraire dans le contexte des transports) entre deux nœuds i et j du graphe est une séquence ordonnée de nœuds* $< u_0, u_1, \ldots, u_k >$ *telle que,*

- $u_k \in N, \forall k \in [0, n]$
- $u_0 = i, u_n = j$
- $u_z, u_{z+1} \in < u_0, u_1, \ldots, u_k >$ *t.q.* $(u_z, u_{z+1}) \in A$

L'origine du chemin (c'est-à-dire le nœud i) est appelé source et la destination (c'est-à-dire le nœud j) est appelé puit.
Si les nœuds composant le chemin sont tous distincts, ce chemin est élémentaire. La longueur du chemin est le nombre d'arcs dans le chemin, c'est à dire k.

Dans un graphe orienté, un chemin $< u_0, u_1, \ldots, u_k >$ forme un circuit si $u_0 = u_k$ et si le chemin comporte au moins un arc ($k \geq 1$). De plus, ce circuit est élémentaire si les sommets $u_0, u_1, \ldots, u_k$ sont tous distincts. Une boucle est un circuit de longueur 1. On retrouve également ces différentes notions de cheminement dans les graphes non orientés. Dans ce cas, on parlera de chaîne au lieu de chemin, et de cycle au lieu de circuit. Un graphe sans cycle est dit acyclique.

Représentation d'un graphe

Il existe deux façons classiques de représenter un graphe : par une matrice d'adjacence ou par un ensemble de listes d'adjacence.

Matrice d'adjacence

Une matrice $\mathscr{M}$ de dimension $n \times n$ représente un graphe de la manière suivante : $\mathscr{M}_{uv} = 1$ s'il existe un arc du nœud u au nœud v et 0 sinon. Plutôt que d'attribuer une valeur booléenne, il est possible d'utiliser le poids de l'arc pour marquer son existence. Si cette représentation permet de tester facilement l'existence d'un arc en $O(1)$, l'obtention de l'ensemble des successeurs d'un noeud est en $O(n)$.

-Taille mémoire nécessaire : la matrice d'adjacence d'un graphe ayant n noeuds nécessite de l'ordre de $O(n^2)$ emplacements mémoire. Si le nombre d'arcs est très inférieur à n^2, cette représentation est donc loin d'être optimale.

Liste d'adjacence

La représentation par listes d'adjacence de G consiste en un tableau T de n listes, c'est à dire une liste pour chaque nœud de l'ensemble N. Pour chaque nœud $u_i \in N$, la liste d'adjacence $T[u_i]$ est une liste chaînée de tous les nœuds u_j, tel qu'il existe un arc ou une arête $(u_i, u_j) \in A$. Autrement dit, $T[u_i]$ contient la liste de tous les nœuds successeurs de u_i. Les nœuds de chaque liste d'adjacence sont généralement chaînés selon un ordre arbitraire.

-Taille mémoire nécessaire : si le graphe G est orienté, la somme des longueurs des listes d'adjacence est égale au nombre d'arcs de A car l'existence d'un arc (u_i, u_j) se traduit par la présence de u_j dans la liste d'adjacence de $T[u_i]$. En revanche, si le graphe n'est pas orienté, la somme des longueurs de toutes les listes d'adjacence est égale à deux fois le nombre d'arêtes du graphe. Puisque (u_i, u_j) est une arête, si u_i appartient à la liste d'adjacence de $T[u_j]$ alors u_j appartient forcément à la liste d'adjacence de $T[u_i]$. Par conséquent, la liste d'adjacence d'un graphe ayant n nœuds et m arcs ou arêtes nécessite de l'ordre de $O(n+m)$ emplacements mémoires. Cette représentation nécessite donc moins de mémoire qu'une matrice d'adjacence et l'obtention de la liste des successeurs d'un noeud est en $O(1)$. Lorsque le degré des noeuds est indépendant de la taille du graphe — comme dans un réseau routier — alors la mémoire nécessaire est en $O(n)$. Cette représentation est donc préférée pour les graphes grands et peu denses.

2.1.2 Algorithmes de plus court chemin

Le problème de plus court chemin sur les graphes a fait l'objet de nombreuses études. Il est devenu l'un des principaux problèmes de la théorie des graphes [Deo and Pang, 1984]. De nombreux algorithmes ont été proposés pour résoudre ce problème [Cherkassky et al., 1996, Gallo and Pallottino, 1988]. Ils se différencient par un certain nombre de caractéristiques, à savoir :

- Poids acceptables : certains algorithmes n'acceptent que des arcs dont le poids est positif.
- Chemins calculés : il existe des algorithmes qui calculent le plus court chemin de noeud à noeud, entre toutes les paires de nœuds ou encore d'un noeud vers tous les autres. Dans ce dernier cas, l'algorithme va générer un arbre couvrant.
- Prise en compte d'informations externes : parfois l'utilisation d'une connaissance externe à la structure du graphe peut accélérer la recherche.

Les deux algorithmes les plus célèbres et les plus utilisés sont les algorithmes proposés par E.W. Dijkstra [Dijkstra, 1959] et Bellman-Ford [Bellman, 1956].

Algorithme de Dijkstra

Pour un nœud source donné, cet algorithme permet de déterminer le plus court chemin entre la source et tous les autres nœuds du graphe (de type un-vers-plusieurs). Cependant, l'algorithme peut être utilisé pour calculer uniquement un plus court chemin entre un nœud source et un nœud

destination (de type un-vers-un). Une particularité de cet algorithme est qu'il ne fonctionne que sur les graphes valués avec des coûts positifs associés aux arcs ou arêtes. Ce dernier est un algorithme dit d'étiquetage (labelling) : une étiquette (correspondant à une distance ou une durée) est associée à chaque noeud $i \in A$ et contient la valeur du coût minimal correspondant au plus court chemin entre le noeud source s et le noeud i. Au début de l'algorithme, toutes les étiquettes sont initialisées à $+\infty$, à l'exception de l'étiquette du noeud s qui prend la valeur 0. La file de priorité Q contient la liste de nœuds à traiter. Initialement, Q contient uniquement la source s. Le principe de l'algorithme de Dijkstra est ensuite de retirer un noeud avec une plus petite étiquette de la file Q, suivie d'une mise à jour pour les étiquettes des noeuds successeurs si la valeur de leur étiquette est améliorée.

L'algorithme de Dijkstra est un algorithme dit de *label-setting* car il respecte la propriété suivante : à chaque fois qu'un noeud i est retiré de la file Q, la valeur de son étiquette est définitive et correspond au plus court chemin de s à i. Les algorithmes de *label-setting* sont opposés aux algorithmes de *label-correcting* qui sont également des algorithmes de labelling mais qui ne respectent pas cette condition. Cependant, si l'objectif initial est de calculer uniquement le plus court chemin entre deux noeuds, comme de s vers t, il est possible d'arrêter l'algorithme dès que le noeud t est retiré de la file Q. En effet, comme il s'agit d'un algorithme de *label-setting*, lorsque t est retiré de la file Q, nous sommes alors certains que son étiquette associée correspond au coût du plus court chemin de s à t.

Complexité. La complexité de l'algorithme dépend de l'implémentation du graphe (par matrice ou par liste d'adjacence), mais aussi de la façon de gérer la file de priorité Q. Si on utilise une matrice d'adjacence, l'algorithme sera en $O(n^2)$. En revanche, si on utilise une liste d'adjacence, alors :

- Si Q est implémenté par une liste linéaire ou un tableau, il faudra chercher à chaque itération, le noeud dans Q ayant la plus petite étiquette. Étant donné qu'il y a n itérations, et qu'au premier passage Q contient n éléments, et qu'à chaque passage suivant Q contient un élément de moins, il faudra au total faire de l'ordre de $n + (n-1) + (n-2) + \cdots + 2 + 1$ opérations, soit $O(n^2)$. En revanche, chaque arc étant relâché une seule fois, les opérations de relâchement prendront de l'ordre de m opérations. Au total on aura donc une complexité en $O(n^2)$.
- Pour améliorer les performances de l'algorithme, il faut trouver une structure de données permettant de trouver rapidement la plus petite étiquette dans la file Q. Pour cela, un tas binaire peut être utilisé. Un tas binaire permet de trouver le plus petit élément d'un ensemble en temps constant (immédiatement). En revanche, l'ajout, la suppression ou la modification d'un élément dans un tas binaire comportant n éléments prendra de l'ordre de $\log(n)$ opérations. Par conséquent, si Q est implémenté avec un tas binaire, on obtient une complexité de $O(m \cdot \log(n))$. Autrement, en utilisant le tas de Fibonacci [Fredman and Tarjan, 1987], la complexité se réduit à $O(m + n \cdot \log(n))$.

Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford se différencie de celui de Dijkstra par la possibilité d'avoir des arcs de coût négatif dans le graphe. Plus exactement, il permet de trouver les plus courts chemins dans le cas où le graphe contient des arcs dont le coût est négatif. Le graphe ne doit pas contenir de circuits absorbants (dans ce cas, l'algorithme de Bellman-Ford va détecter l'existence de circuits négatifs). Si un tel circuit absorbant est détecté, l'algorithme s'arrête. Contrairement à l'algorithme de Dijkstra qui sélectionne les nœuds un par un et étend une étiquette aux nœuds successeurs, l'algorithme de Bellman-Ford étend à chaque itération l'ensemble des étiquettes en utilisant tous les arcs. Comme le plus court chemin jusqu'au noeud peut être corrigé à chaque itération, on parle d'algorithme de type *label-correcting*. Étant donné que le graphe ne comporte pas de circuits absorbants, un plus court chemin est nécessairement élémentaire. Donc, l'algorithme itère au plus n fois ce qui correspond à

une borne maximale du nombre de nœuds composant le plus court chemin.

Complexité. Dans cet algorithme, chaque arc sera relâché $n-1$ fois, et on effectuera donc au total $(n-1)\cdot m$ relâchements successifs. Si le graphe est représenté par des matrices d'adjacence, on aura une complexité en $O(n^3)$, alors que s'il est représenté par des listes d'adjacence, on aura une complexité en $O(n\cdot m)$. En pratique, on pourra arrêter l'algorithme dès lors qu'aucune étiquette n'a été modifiée pendant une itération complète. On pourra aussi mémoriser à chaque itération, l'ensemble des nœuds pour lesquels la valeur de l'étiquette a changé afin de ne relâcher que les arcs partant de ces sommets lors de l'itération suivante .

2.1.3 Techniques d'accélération pour les algorithmes de plus court chemin

L'étude présentée dans [Sanders and Schultes, 2007] démontre la sensibilité de l'algorithme de Dijkstra pour des réseaux de grande taille. Pour y remédier, plusieurs techniques d'accélération simple ont été développées : la recherche bidirectionnelle et la recherche guidée $A^\star$.
En se basant sur l'algorithme de Dijkstra, la recherche bidirectionnelle et la recherche guidée $A^\star$, un certain nombre de techniques de haute performance pré-traite le graphe d'entrée statique en amont pour obtenir des accélérations drastiques lors de l'arrivée d'une requête. La Contraction Hiérarchique (CH) [Geisberger et al., 2012] est une technique d'accélération ayant un compromis idéal entre le pré-traitement et le traitement immédiat lors d'une arrivée d'une requête en termes de temps de calcul et d'espace de stockage. Un réseau routier avec des millions de nœuds et d'arcs peut être pré-traité en quelques minutes tandis que les requêtes sont exécutées en centaines de microsecondes. Transit Node Routing (TNR) [Arz et al., 2013] est l'une des techniques les plus rapides pour le calcul des plus courtes distances dans un réseau routier. En pré-traitant le réseau routier d'entrée, les requêtes sont traitées en temps constant avec des recherches rapides dans la table des distances.

Recherche bidirectionnelle

Une idée très simple pour améliorer l'algorithme de Dijkstra est d'effectuer en parallèle deux recherches d'itinéraire, l'une en partant de la source sur le graphe original (ayant comme file de priorité Q_F) et l'autre en partant de la destination sur le graphe inversé (ayant comme file de priorité Q_B). Lorsque les deux recherches ont toutes les deux visitées un même noeud, le premier coût provisoire du plus court chemin depuis la source vers la destination est connu. L'intersection des deux recherches dans un nœud ne garantit pas l'optimalité du plus court chemin trouvé. Pour garantir l'optimalité, les deux recherches doivent se poursuivre jusqu'à ce que le coût provisoire minimal des deux files de priorité soit supérieur au coût du plus court chemin provisoire (ce qui correspondra enfin au plus court chemin).

La figure 2.3 illustre les différents espaces visités par une recherche simple de l'algorithme de Dijkstra et une recherche bidirectionnelle.

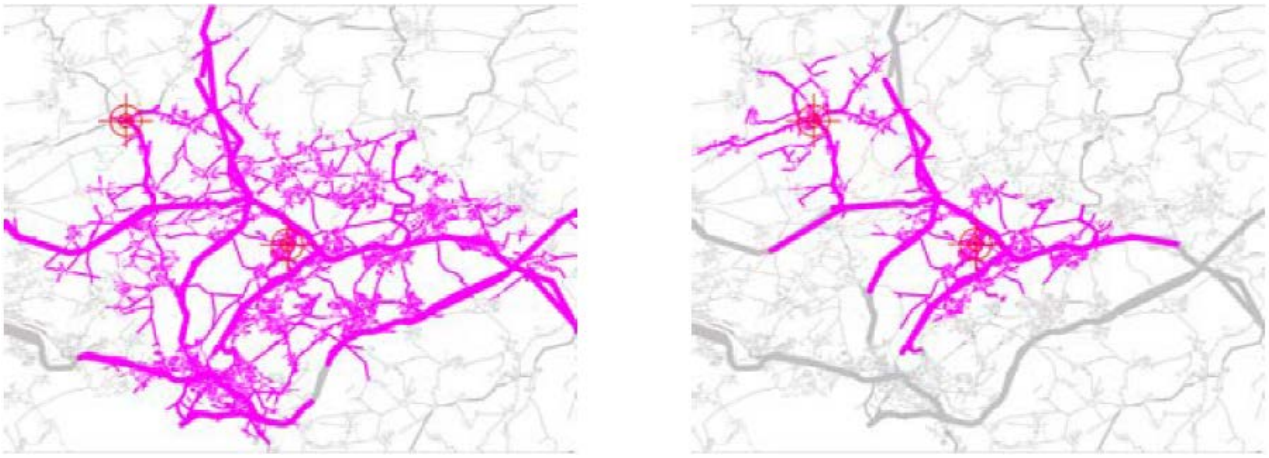


FIGURE 2.3 – L'espace de recherche de l'algorithme de Dijkstra sur un réseau routier entre une source et une destination. A gauche : la version simple de l'algorithme de Dijkstra. A droite : la version bidirectionnelle de Dijkstra.

Techniques de Recherche-guidée

L'algorithme de Dijkstra parcourt tous les nœuds avec des étiquettes plus petites à l'étiquette du noeud de la destination t. En revanche, les techniques de recherche-guidée ("goal-directed") visent à «guider» la recherche vers la destination en évitant les visites des nœuds ou des arcs qui ne sont pas dans la direction de la destination t. Ces dernières exploitent généralement la topologie géométrique ou les propriétés du réseau lui-même.

Recherche $A^\star$. La recherche $A^\star$ est la méthode la plus classique des goal-directed. Cette dernière a été introduite par [Hart et al., 1968]. Contrairement à l'algorithme de Dijkstra classique, la recherche dans $A^\star$ est guidée par une estimation de plus court chemin depuis chaque noeud vers la destination. Cette estimation n'est autre que la borne inférieure du coût du plus court chemin jusqu'au noeud destination. Si cette heuristique sous-estime la distance réelle, alors l'algorithme est optimal. L'espace de recherche associé à cette approche correspond généralement au demi-disque autour de la source (plus exactement le demi-disque situé à côté du noeud destination). Dans un réseau routier, il est habituel d'utiliser la distance euclidienne ou la distance à vol d'oiseau comme métrique pour estimer la distance restante entre un noeud courant et le noeud destination.
De meilleures bornes inférieures peuvent être obtenues en utilisant l'approche ALT qui est basée sur la recherche $A^\star$, *landmarks* et l'inégalité triangulaire [Goldberg and Harrelson, 2005]. Dans cette approche, l'inégalité triangulaire est basé sur les plus courtes distances réelles dans le graphe et non pas selon les estimations euclidiennes ou d'autres métriques d'estimation. Durant la phase du pré-traitement, un petit sous-ensemble de nœuds-repères L est sélectionné. Ces noeuds-repères sont appelés des *landmarks*. Cette méthode stocke également les distances entre chaque paire de noeuds dans L ainsi que leur distance vers tous les autres noeuds du graphe. Lors du traitement d'une requête, des inégalités triangulaires impliquant des nœuds-repères pour calculer des bornes inférieures valides pour tout noeud u sont utilisées. Plus spécifiquement, étant donné un ensemble de *landmarks* $l \in L$, les distances $dist(l,u)$ et $dist(u,l)$ pour chaque noeud u dans A sont déjà connues. Ainsi, pour un landmark donné $l \in L$, les inégalités suivantes sont respectées :

$$dist(l,u) + dist(u,v) \geq dist(l,v) \text{ et } dist(u,v) + dist(v,l) \geq dist(u,l)$$

Par conséquent, $\underline{dist}(u,v) := \max_{l \in L} \max\{d(l,v) - d(l,u), dist(u,l) - d(v,l)\}$ fournit une borne inférieure valide pour la distance $dist(u,v)$. Voir la figure 2.4 pour une illustration.

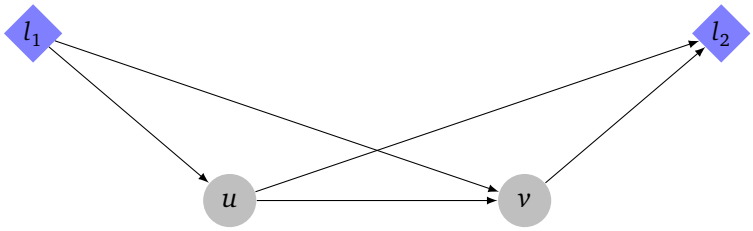


FIGURE 2.4 – Les inégalités triangulaires pour les deux repères l_1 et l_2.

La qualité de la borne inférieure (et donc les performances des requêtes) dépend fortement des noeuds choisis comme points de repère lors du pré-traitement. Sur les réseaux routiers, la sélection des noeuds-repères bien espacés entre eux et qui se trouvent généralement à proximité des frontières du graphe conduit à de meilleurs résultats, avec des temps d'exécution des requêtes acceptables en moyenne [Efentakis and Pfoser, 2013, Goldberg and Werneck, 2005].
La figure 2.5 montre les zones de recherche effectuées par trois types d'algorithmes différents dans une grille carrée représentant la zone de Manhattan.

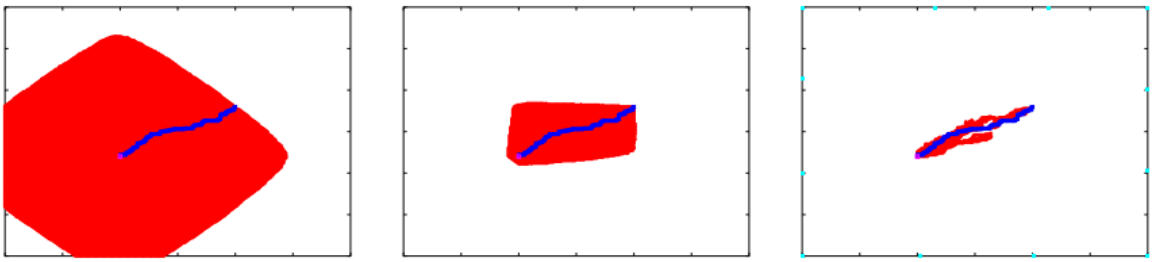


FIGURE 2.5 – Le plus court chemin est représenté par le trait bleu. L'algorithme de Dijkstra effectue une grande recherche autour de la source (figure à gauche). La figure au milieu représente la zone de recherche effectuée par la méthode $A^\star$, alors que la figure de droite représente la zone de recherche explorée par la méthode ALT.

Une technique possible pour améliorer la méthode bidirectionnelle est de guider les deux recherches (par $A^\star$ ou ALT), la première recherche provenant de la source sera guidée vers la cible, et inversement la recherche provenant de la cible sera guidée vers la source.

Conteneurs géométriques La méthode dite *Conteneurs géométriques.* détermine pour chaque arc $a = (u,v) \in A$, une étiquette $L(a)$ qui stocke l'ensemble des nœuds V_a dans laquelle un plus court chemin à partir de u commence par l'arc a. Au lieu de stocker explicitement l'ensemble V_a, $L(a)$ approxime cet ensemble en utilisant les informations géométriques (c'est-à-dire, les coordonnées géographiques) des noeuds dans V_a. Les auteurs dans [Schulz et al., 2000] approximent l'ensemble V_a par un secteur angulaire (centré en u) qui couvre tous les noeuds dans V_a. Dans [Wagner et al., 2005], les auteurs proposent d'autres formes géométriques, telles que des ellipses et des enveloppes convexes en montrant également leur bon fonctionnement. En effet, stocker tous les ensembles associés à chaque arc exigerait un espace de $O(nm)$, au lieu de cela ils stockent à la place un objet géométrique

(appelé conteneur) pour chaque arc contenant au moins un noeud dans l'étiquette. Ce qui explique que l'espace de stockage est réduit à $O(n+m)$.
Cependant, lorsqu'une requête arrive dans le système, un algorithme de Dijkstra modifié est exécuté en ignorant chaque arc a dans lequel la destination t n'est pas dans l'étiquette $L(a)$. Autrement dit, les arcs visités seront limités aux arcs pour lesquels le noeud destination est dans l'étiquette associée. Une version dynamique du problème est également présentée dans [Wagner et al., 2005]. Les auteurs ont présenté des algorithmes qui maintiennent dynamiquement les conteneurs géométriques lorsque le poids des arcs augmentent ou diminuent en gérant également les suppressions et les insertions des noeuds dans le graphe. Un inconvénient de cette approche est que son prétraitement nécessite essentiellement un ensemble de paires de calculs de plus court chemin, ce qui est coûteux en temps de calcul.

Marquage d'arcs L'approche *Marquage d'arcs.* [Lauther, 2006],[Hilger et al., 2009] est assez semblable à l'approche *Conteneurs géométriques* dans le sens où les deux approches visent à éviter le parcours des arcs inutiles. Cependant, contrairement à l'approche *Conteneurs géométriques*, cette dernière est basée sur le partitionnement d'un graphe en k régions qui sont à peu près équilibrées en termes de nombre d'arcs. Chaque arc maintient un vecteur booléen de longueur k (représentant les k marqueurs des régions), où chaque marque indique si le plus court chemin vers la région correspondante à la destination passe par cet arc. Le vecteur est ordonné de tel manière qu'un élément se trouvant à la position i correspond à la région i. Ainsi, lors de la recherche d'itinéraires, les arcs ne contenant pas la région correspondante à la destination de la requête sont ignorés, et la zone de recherche est orientée vers la destination. La combinaison de cette approche avec une recherche bidirectionnelle améliore les performances en termes de temps d'exécution des requêtes.

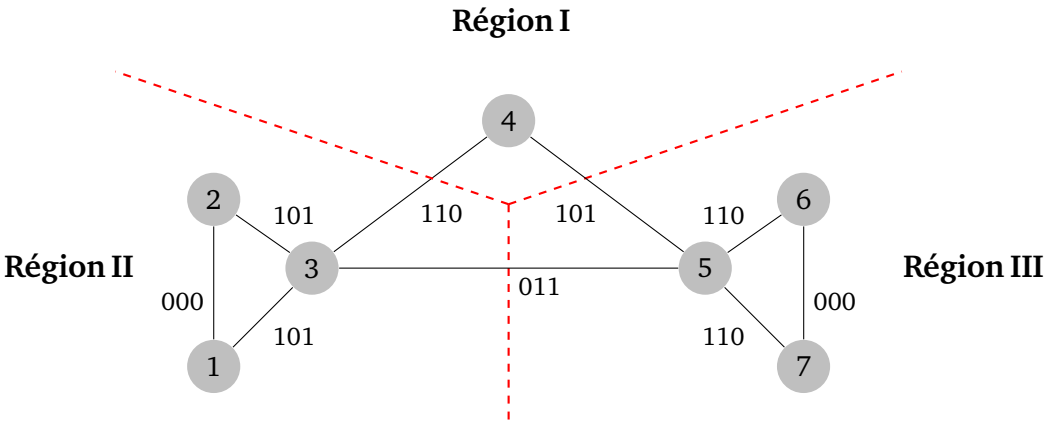


FIGURE 2.6 – Exemple de partitionnement du graphe pour l'approche marquage d'arcs. Le premier bit à 1 veut dire qu'il existe un chemin qui emprunte l'arc pour atteindre la région I, le deuxième bit la région II et le troisième bit la région III.

Méthodes Hiérarchiques

Les méthodes hiérarchiques visent à exploiter la hiérarchie inhérente des réseaux routiers selon les différentes catégories de route. Les trajets longues distances finissent par converger vers un sous réseau composé de routes importantes, telles que les *autoroutes*, les voies rapides. Intuitivement, une fois que la recherche du plus court chemin est loin de la source et de la destination, il suffit

seulement de parcourir les sommets de ce sous-réseau. Ces approches sont très efficaces à condition que la source et la cible soient assez éloignées et que le réseau autoroutier soit bien développé dans la zone concernée.

Highway Hierarchies. Avant la *Contraction hiérarchies* [Geisberger et al., 2012], les auteurs développent le concept de *Highway Hierarchies* [Schultes, 2005, Sanders and Schultes, 2006]. Ce concept est basé sur l'observation suivante : dans un réseau routier, un niveau hiérarchique peut être attribué à chaque arc. Les arcs correspondants à des routes non principales obtiennent un niveau hiérarchique faible, tandis que les arcs correspondants aux routes principales et rapides obtiennent un niveau hiérarchique élevé. Autrement dit, en dehors de certaines *zones locales* autour de la source et de la destination, seul un sous-ensemble d'arcs "important" est pris en compte afin de trouver le plus court chemin. Cette approche ne classe pas les nœuds mais plutôt les arcs. La notion d'une *zone locale* est formalisée par la définition d'un ensemble de nœuds de voisinage pour chaque noeud du graphe. Ils réalisent d'abord une recherche locale autour de s et de t en envisageant toutes les routes possibles (par exemple, autour de 20 km), puis ils passent à la recherche dans un réseau plus petit que le graphe original en envisageant uniquement les routes nationales et autoroutes (par exemple autour de 100 km) et ensuite ils passent à une hiérarchie supérieure en considérant uniquement les autoroutes.
Une hiérarchie sur le graphe G se compose de plusieurs niveaux $G_0, G_1, G_2, ..., G_L$. Niveau 0 correspond au graphe original G. Le niveau 1 est obtenu en construisant un *réseau autoroutier* sur le graphe de niveau 0, le niveau 2 en construisant un *réseau autoroutier* sur le graphe du niveau 1 et ainsi de suite. La requête est exécutée en utilisant une adaptation de l'algorithme de Dijkstra bidirectionnelle sur le graphe G à plusieurs niveaux. La recherche commence de s et de e au niveau 0 (graphe complet) en effectuant une recherche locale dans le graphe d'origine (niveau 0). Ensuite, la recherche s'oriente vers le *réseau autoroutier* (niveau 1) en effectuant une recherche locale sur ce *réseau autoroutier*. Enfin, elle passe au prochain niveau de la hiérarchie de la route, et ainsi de suite.
La figure 2.7 donne une représentation schématique de l'espace de recherche.

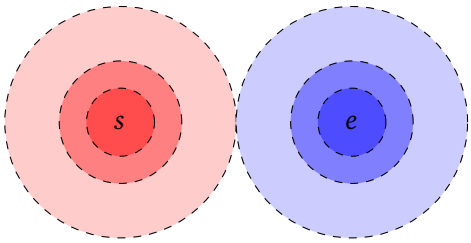


FIGURE 2.7 – Représentation schématique de la recherche au niveau 0 (couleur sombre), au niveau 1 (couleur claire) et au niveau 2 (couleur très claire) d'une hiérarchie d'autoroute.

La figue 2.8 représente un exemple réel.

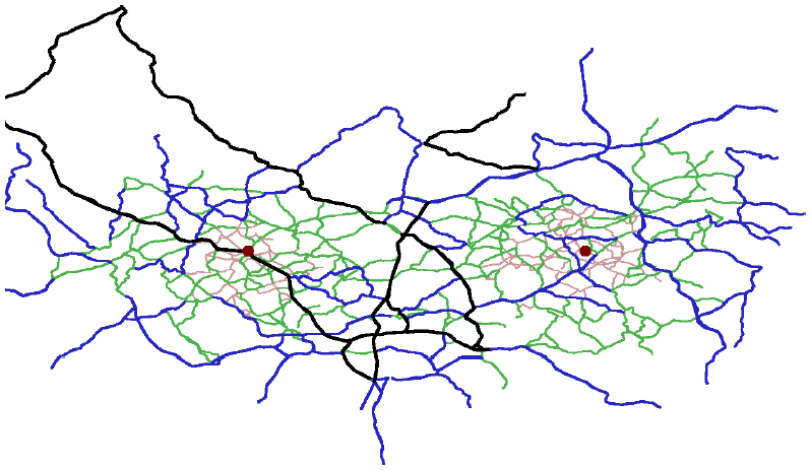


FIGURE 2.8 – Espace de recherche pour une requête de Limburg (ville allemande). La source et la cible sont représentées par les deux cercles. Le cercle à gauche correspond au lieu de départ tandis que le cercle à droite correspond à la destination. Ce dernier se trouve aux environs de 100 km du point de départ. Les lignes les plus épaisses correspondent au niveau supérieur. Il est à noter que les arcs représentant les longs sous-chemins ne sont pas tracés comme des raccourcis directs, mais plutôt par des itinéraires géographiques réels.

Dans cette approche, les modifications concernant les changements structurels hiérarchiques ne se mettent pas facilement à jour. Autrement dit, une hypothèse clé pour cette dernière est que le graphe doit être statique et ne pas changer au fil du temps. Toutefois, cela est faux pour certaines applications. Une solution à ce problème est présentée dans [Schultes and Sanders, 2007] sous le nom *Dynamic Highway-Node Routing*. Les auteurs introduisent une technique dynamique pour la planification d'itinéraires rapides dans les grands réseaux routiers. Ils gèrent également les scénarios qui se présentent dans les systèmes de navigation suite au changement de poids des arcs (par exemple, en raison d'un embouteillage). Le temps de mise à jour des informations pré-traitées varie de 2 à 40 ms permettant des traitements de requêtes rapides d'une milliseconde en moyenne. Ces approches sont efficaces pour la recherche des plus courts chemins en temps mais pas forcément pour les plus courts chemins en distance.

Transit-Node Routing. *Transit-Node Routing* (TNR) [Arz et al., 2013, Bast et al., 2009] repose sur l'observation "humaine" que certains nœuds sont utilisés pour de nombreuses routes et d'autres très rarement (presque tous les longs trajets *passent par le périphérique* quelque soit le lieu de départ et le lieu d'arrivée). Cette dernière utilise la technique des tables des distances sur un sous-ensemble de noeuds $T \in A$. Ce sous-ensemble de noeuds, appelés noeuds de transit, pour lesquels le plus court chemin de n'importe quelle paire de noeuds suffisamment éloignés, passe au moins par un des noeuds de cet ensemble T. Pour chaque noeud $u \in A/T$, un ensemble pertinent de *nœuds d'accès* $NA(u) \in T$ lui est associé. Un noeud de transit $v \in T$ est un *noeud d'accès* pour le noeud u si et seulement si il existe un plus court chemin P dans G à partir du noeud u, tel que v est le premier noeud de transit dans le chemin P. En plus du noeud lui-même, le pré-traitement stocke également les distances entre u et ses nœuds d'accès. Le nombre de noeuds d'accès est généralement faible dans les réseaux routiers. Pendant la phase du pré-traitement, la méthode calcule également les distances entre tous les noeuds de transit. Ce prétraitement permet alors de déterminer rapidement un plus court chemin en utilisant uniquement les distances stockées dans la table des distances. Plus spécifiquement, le

chemin sélectionné est celui qui minimise la distance combinée $s \rightarrow NA(s) \rightarrow NA(t) \rightarrow t$.
La figure 2.9 illustre un exemple d'une requête TNR.

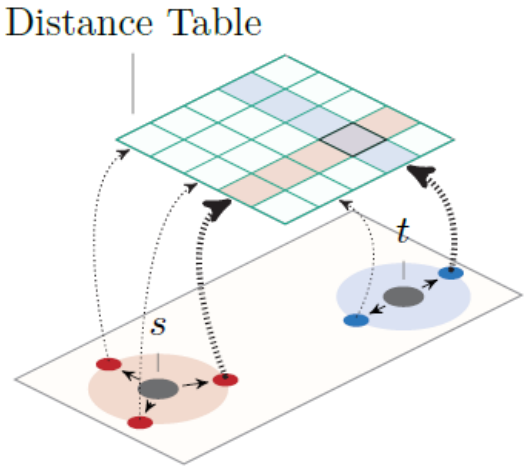


FIGURE 2.9 – Illustration d'une requête TNR, entre une source s et une destination t. Les points rouges (respectivement bleus) sont les noeuds d'accès de s (respectivement t). Les flèches indiquent les lignes/colonnes respectives de la table des distances. Les entrées mises en évidence correspondent à des noeuds d'accès qui réduisent au maximum la distance combinée $s \rightarrow NA(s) \rightarrow NA(t) \rightarrow t$.

Contraction Hiérarchique. Une approche importante pour exploiter la hiérarchie est d'utiliser des raccourcis. L'idée est d'augmenter le graphe G avec des raccourcis afin qu'ils puissent être utilisés par les requêtes de longue distance en ignorant les nœuds «sans importance» [Geisberger et al., 2012]. Plus spécifiquement, cette dernière consiste à supprimer un noeud v et insérer les arcs nécessaires, c'est à dire des *raccourcis* entre voisins, de telle sorte que leurs distances par paires soient conservées. Cette opération est appelée *contraction* du noeud v. La figure 2.10 montre un exemple de contraction d'un noeud.

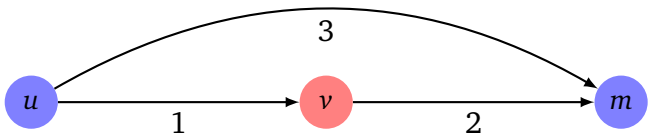


FIGURE 2.10 – Le graphe avant et après la contraction du noeud v. A noter que si l'arc (u, m) n'existait pas pendant l'opération de la contraction v, cet arc serait inséré comme un raccourci.

Tous les nœuds sont contractés dans un ordre spécifique, appelé *ordre de contraction*. Cet ordre est déterminé en utilisant des heuristiques dans le but de mettre les nœuds correspondants à des rues non principales en bas niveau, et les nœuds correspondants aux voies rapides dans la partie supérieure. L'objectif est d'insérer le moins possible de raccourcis dans le graphe. Une heuristique simple pour définir l'ordre de contraction est de considérer le nombre de raccourcis à ajouter moins le nombre d'arcs supprimés (si un nœud est contracté directement). Ces valeurs sont appelées *arcs de différence*. L'ordre est alors défini en fonction de l'augmentation d'*arcs de différence*. On

peut également sélectionner le noeud en tenant compte d'une combinaison de plusieurs facteurs, à savoir le nombre de raccourcis supplémentaires et le nombre de noeuds à proximité déjà contractés [Geisberger et al., 2012, Kieritz et al., 2010].
Dans la pratique, la contraction de nœuds ne se résume pas à leur suppression du graphe. Au lieu de cela, pendant le processus itératif de contraction, tous les nœuds préalablement contractés sont temporairement ignorés. Une requête d'un plus court chemin de s à t est effectuée sur le graphe contracté G', en utilisant un algorithme de Dijkstra bidirectionnel modifié en visitant uniquement les arcs menant à des noeuds d'un niveau d'importance supérieur (voir la figure 2.11). Plus spéci-

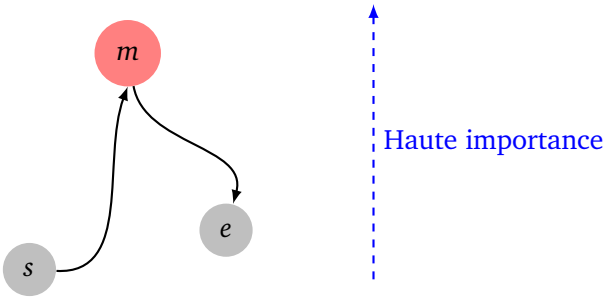


FIGURE 2.11 – Illustration d'une requête visitant le nœud de plus haute importance m.

fiquement, à partir du noeud source nous ne considérons que les arcs sortant vers les nœuds avec un ordre élevé. De même, pour la recherche arrière à partir du noeud cible, seuls les arcs vers les nœuds avec un ordre élevé sont considérés. L'intersection des nœuds établis en provenance de la source et de la destination donne la distance minimale entre la source et la cible. Les auteurs démontrent que pour les grands réseaux comme l'ensemble du réseau routier de l'Europe occidentale, le temps d'exécution d'une requête est de l'ordre de quelques millisecondes. Pour plus de détails, voir les travaux dans [Geisberger et al., 2012]. La *Contraction Hiérarchique* (CH) est une succession de la méthode *"Highway hierarchies*" [Schultes, 2005, Sanders and Schultes, 2006] et Highway Node Routing [Schultes and Sanders, 2007] qui sont basées sur des idées similaires. *CH* est non seulement plus rapide, mais aussi conceptuellement plus simple.

2.1.4 Plus court chemin sur les graphes dynamiques

Les transport en commun fonctionnent sur des horaires fixes, ce qui n'était pas forcement le cas dans les réseaux routiers. En revanche, dans les réseaux routiers avancés, une modélisation plus adéquate correspondrait aux graphes dynamiques. Ces derniers permettent d'intégrer les temps d'attente pour les feux rouges, les perturbations au niveau du trafic, etc. Il existe trois approches principales pour modéliser un réseau de transport en commun : le modèle *condensé*, le *Time-expanded graph*, et le *Time-depended graph*. La principale différence entre les modèles, le *Time-expanded* et le *Time-depended* est due à la pondération attribuée aux arcs du graphe qui est constante dans un cas et ne l'est pas dans l'autre cas.

Timetable

La base de chaque modèle est la *Timetable* (calendrier) à partir de laquelle nous construisons le graphe qui permet de calculer les plus courts chemins. *Timetable* est un tableau de 4-uplets (B, Z, P, C) avec B un ensemble de stations où les véhicules peuvent effectuer des arrêts (arrêts de bus, de métro et de train, etc), Z un ensemble de véhicules, P la périodicité du calendrier et C qui correspond

à l'ensemble des connexions élémentaires. Une connexion élémentaire est définie par un 5-uplets $c = (z, s_1, s_2, \tau_1, \tau_2)$. Un véhicule $z \in Z$ se déplace à l'instant τ_1 de la station $s_1 \in B$ vers la station $s_2 \in B$ avec une heure d'arrivée τ_2 telle que $\tau_1 < P$ et $\tau_2 < P$. Il est à noter que pour une connexion élémentaire, le véhicule ne doit effectuer aucun arrêt entre les deux arrêts s_1 et s_2. Ainsi, un itinéraire est composé de plusieurs connexions élémentaires dans la *Timetable*.
Concernant le calcul du temps de trajet d'une connexion élémentaire c, si l'heure d'arrivée τ_2 est supérieure à l'heure de départ τ_1, alors le temps du trajet correspond à la différence $\tau_2 - \tau_1$. Par conséquent, à cause de la périodicité de la *Timetable*, il est également possible d'avoir un départ dans la soirée et une arrivée au cours du prochain jour . Dans ce cas, $\tau_2 < \tau_1$ et le temps d'arrivée est composé de la durée restante depuis τ_1 jusqu'à minuit, auquel s'ajoute la durée de minuit jusqu'à τ_2. En ce qui concerne la périodicité P, on obtient donc pour la durée du voyage :

$$\Delta(\tau_1, \tau_2) := \begin{cases} \tau_2 - \tau_1 & \text{si } \tau_2 \geq \tau_1 \quad (2.1a) \\ P - \tau_1 + \tau_2 & \text{sinon} \quad (2.1b) \end{cases}$$

La fonction Δ peut être utilisée pour calculer les durées de déplacement entre deux points quelconques.

Modèle Condensé

Le modèle le plus basique pour modéliser un réseau de transport en commun est le modèle *condensé*. Il représente la structure du réseau sans les horaires correspondant, il est cependant indépendant du temps. Pour chaque station $s \in B$, il existe seulement un noeud $v \in V$ dans le graphe. Un arc $a = (u, v)$ est introduit dans le graphe si et seulement si il existe au moins une connexion élémentaire dans la *Timetable* (calendrier) reliant la station u à la station v. La durée associée a un arc $a = (u, v)$ est fixée à la durée minimale de toutes les connexions élémentaires existantes allant de u à v. C'est-à-dire,

$$\Delta(a) := \min_{c \in C, u = s_1(c), v = s_2(c)} \Delta(\tau_1(c), \tau_2(c))$$

Généralement ces modèles condensés correspondent aux cartes représentant le réseau du métro ou des bus qui indiquent seulement les stations, les connexions et les lignes, et non pas les heures de départ. Malgré le fait que le modèle condensé génère un petit graphe et permet de représenter adéquatement la structure du réseau, une requête de plus court chemin est calculée uniquement en se basant sur les bornes inférieures associées à chaque connexion élémentaire existante dans le réseau. Ainsi, pour calculer les temps de déplacement exacts, ce modèle ne peut pas être appliqué. En revanche, dans le cas où la fréquence des différents transports en commun est très élevée, ce type de modélisation donne des résultats satisfaisants.

Modèle Time-expanded

La raison pour laquelle le modèle condensé n'est pas utile pour la détermination des durées exactes des plus courts chemins, est qu'il ne tient pas compte des horaires de départ et d'arrivée de la *Timetable*. Pour résoudre ce problème tout en étant capable d'utiliser une approche indépendante du temps, le modèle *Time-expanded* a été développé. Le graphe généré est cependant nommé graphe *espace-temps*.
Historiquement, il y a eu deux versions : la version simple du modèle de *Time-expanded* [Schulz, 2005], appelé *Simple Time-expanded graph*. Cette dernière ne tient pas compte des durées réelles de transfert effectué entre deux véhicules dans les stations. Pour intégrer cet aspect, le modèle simple a été amélioré par un modèle connu sous le nom *Realist Time-expanded graph*.

Version Simple Les nœuds dans le graphe ne correspondent plus aux stations dans la *Timetable* (calendrier), mais plutôt à des événements. Dans la version simple du modèle à *Time-expanded*, il y a deux types d'événements : les événements de départ et les évènements d'arrivée. Pour chaque connexion élémentaire $c = (Z, s_1, s_2, \tau_1, \tau_2)$ deux noeuds sont créés : un noeud v de départ, qui représente l'événement de départ du véhicule z de la station s_1 à l'instant τ_1, et un noeud d'arrivée u qui représente l'évènement d'arrivée du véhicule z à la station s_2 à l'instant τ_2.
En plus des nœuds représentants les événements de départ et d'arrivée, deux types d'arcs sont insérés. Pour deux événements (nœuds) appartenants à la même connexion élémentaire c, un arc sortant de l'événement de départ vers l'événement d'arrivée est inséré. Le poids de l'arc est défini comme étant la durée du trajet $\Delta(\tau_1(c), \tau_2(c))$. Afin de permettre les transferts au sein des nœuds (événements) appartenant à la même station de s, les nœuds sont classés par ordre croissant par rapport à leur timestamp $(v_0, \ldots, v_k)$. Ainsi, pour deux événements qui se suivent v_i, v_{i+1} au sein d'une même station ayant un timestamp t_i, t_{i+1}, un arc de transfert $a := (v_i, v_{i+1})$ avec une durée $\Delta(\tau_i, \tau_{i+1})$ est inséré dans le graphe. Enfin, pour permettre un transfert après minuit, un arc sortant du dernier noeud de la station v_k vers le premier noeud v_0 avec une durée du trajet $\Delta(v_k, v_0)$ est inséré.

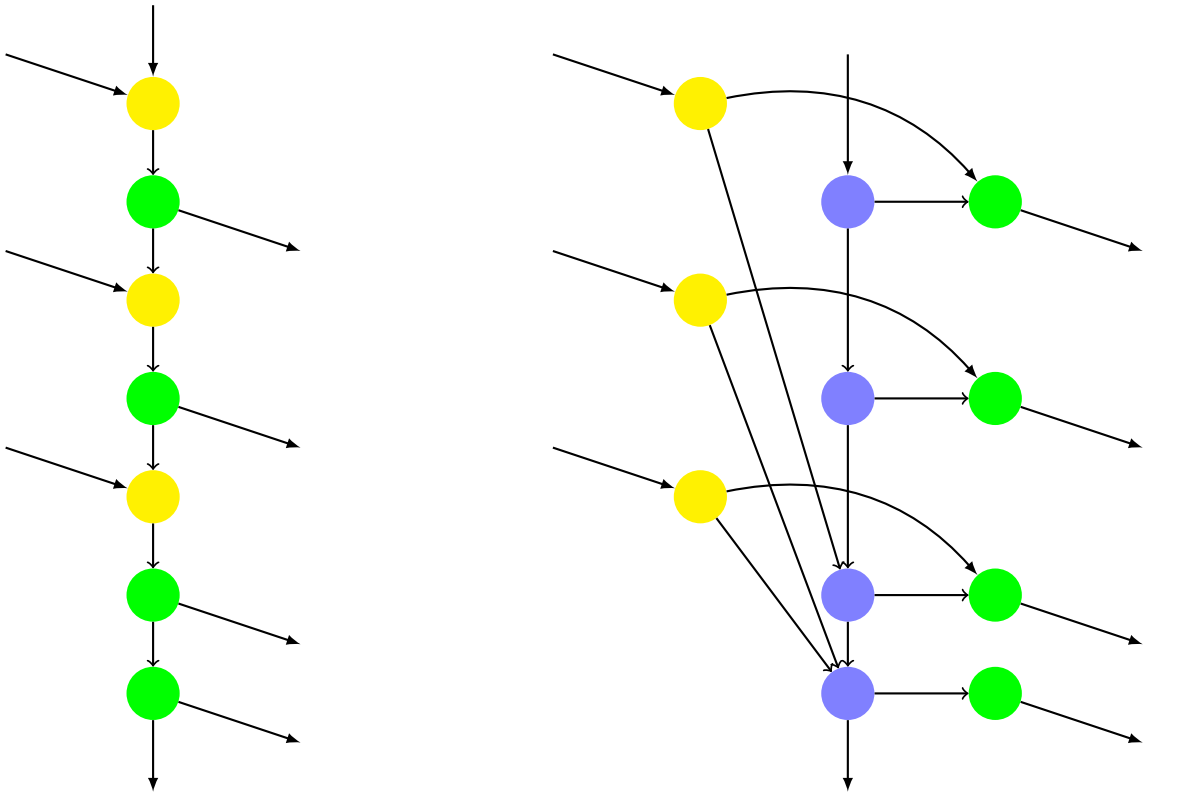
Version Réaliste Dans la version réaliste, la version simple est améliorée en une meilleure modélisation par l'insertion de nœuds et d'arcs de transfert tout en incluant les temps de transfert dans les stations. Pour faire face à des transferts réalistes, la *Timetable C* est prolongée par une fonction de transfert qui ajoute un temps de transfert à chaque station dans le calendrier. Cependant, ce modèle réaliste contient trois types de nœuds : le noeud d'arrivée, le noeud de départ et le noeud de transfert. L'insertion des noeuds de transferts ainsi que les arcs associés augmentent considérablement la taille du graphe. Dans [Antsfeld and Walsh, 2012], les auteurs indiquent qu'au lieu d'insérer les nœuds de transfert et les arcs associés, relier directement les noeuds d'arrivée aux noeuds de départ permet de réduire l'espace de stockage d'environ 30%. La figure 2.12 présente les deux versions d'un modèle *Time-expanded* appliqué sur une station donnée.

Alors que le plus grand avantage du modèle *Time-expanded* est son adaptation facile à l'algorithme de Dijkstra traditionnel, il existe cependant de nombreux inconvénients. Tout d'abord, l'heure d'arrivée au noeud destination est inconnu. Ceci complique l'utilisation de certaines techniques d'accélération, comme la recherche bidirectionnelle. En outre, la taille du graphe devient extrêmement grande, consomme beaucoup de mémoire et conduit également à de très grands espaces de recherche pour l'algorithme de Dijkstra. Par exemple, le réseau de transport public local de Berlin-Brandebourg est composé d'environ 4 millions d'événements de départ et d'arrivée. En l'extrapolant à l'ensemble de l'Europe, ceci donnerait un graphe avec des centaines de millions de nœuds.

Modèle Time-depended

Le modèle *Time-depended* permet de surmonter quelques inconvénients du modèle *Time-expanded* sur la taille du graphe, en introduisant en contre-partie une pénalité de dépendance de temps dans le graphe. Ce modèle modifie les fonctions de coût des arcs, c'est à dire, plutôt que de correspondre à la durée de parcours, les fonctions renvoient l'instant d'arrivée au plus tôt au noeud cible de la connexion élémentaire (des arcs). De même, il existe deux versions du modèle *Time-depended*. La version simple qui ne respecte pas le critère de temps de transfert minimum, alors que la version réaliste traite cette question en renforçant le modèle simple.

Version Simple La version simple est une amélioration du *modèle condensé* introduit précédemment. De même, l'ensemble de nœuds correspondant à l'ensemble des stations et un arc de connexion



(a) Version Simple

(b) Version Réaliste

FIGURE 2.12 – Une même station modélisée par les deux versions du modèle *Time-expanded*. Les nœuds d'arrivées sont représentés en couleur jaune, les nœuds de départ en vert et les noeuds de transfert en violet. Alors que dans la version simple du modèle, le deuxième train peut être atteint à partir du premier, cela est impossible dans le modèle réaliste.

entre deux nœuds u et v est inséré si et seulement si il existe au moins une connexion de u à v dans la *Timetable* (calendrier). Cependant, au lieu d'utiliser des bornes inférieures comme le poids des arcs, leurs poids devient dépendant du temps.

Version Réaliste Comme dans le modèle réaliste de *Time-expanded*, la *Timetable* C est prolongée par une fonction de transfert qui permet d'ajouter un temps de transfert à chaque station dans le calendrier.

Contrainte FIFO Le modèle *Time-depended* a une restriction par rapport au modèle *Time-expanded*. Plus exactement, la résolution du problème "*Earliest Arrival Problem*" en utilisant le modèle *Time-depended* devient NP-difficile si la fonction de coût des arcs du graphe ne satisfait pas la propriété FIFO (*First In, First Out*) (voir [Orda and Rom, 1990]). Cette dernière appelée également "la propriété de non-dépassement" stipule qu'en empruntant un arc (u, v) au plus tôt, on garantit une arrivée au plus tôt en v. Cela signifie que le dépassement des bus/trains sur la même voie n'est pas autorisé. Dans le cas ou un train/bus peut dépasser un autre train/bus partant tout les deux d'une même station (ou de quai pour les trains), la propriété FIFO peut être assurée par une duplication de la route

en attribuant le train/bus lent à une voie, et le train/bus rapide à l'autre. Ce cas ne se produit que rarement dans les systèmes de transport public, donc l'augmentation de la taille du graphe n'est pas nécessaire.

Comparaison des deux modèles

Les auteurs dans [Pyrga et al., 2008] donnent une description détaillée des deux approches en évaluant leur performance. Ces derniers présentent également des algorithmes basiques de planification d'itinéraire. Ils concluent que l'approche *Time-depended* permet un traitement plus rapide des requêtes avec un facteur supérieur à 10, mais cette différence disparaît dès que des caractéristiques réalistes sont prises en compte comme les temps de transfert. Cependant, le modèle *Time-expanded* graphe est plus robuste pour la modélisation des scénarios plus complexes (la prise en compte du temps nécessaire pour changer de quai par exemple).

Plusieurs techniques d'accélération du modèle *Timpe-expanded* ont été présentées dans [Delling et al., 2009]. Avec la présence de la *Timetable*, les techniques d'accélérations dans les réseaux de transport public sont plus complexes que celles utilisées dans les réseaux routiers. Plus spécifiquement, les techniques d'accélération qui se sont avérées très efficaces sur les réseaux routiers n'ont pas donné les mêmes résultats sur les réseaux de transport public [Bast, 2009, Bast et al., 2010, Bauer et al., 2011, Delling et al., 2009, Geisberger, 2010, Pyrga et al., 2008]. Une des différences majeures est l'absence de hiérarchie dans les réseaux de transport public [Bast, 2009].

2.1.5 Modélisation d'un réseau de transport multimodal

La multimodalité permet d'envisager simultanément différentes chaînes de transport (transport en commun, marche à pied, vélo, voiture, etc). Cette dernière a conduit à de nombreux développements que ce soit pour la modélisation ou pour la résolution de problèmes de calcul de chemins. Deux modélisations principales ont été proposées : les graphes multi-valués ou les graphes multi-couches. La modélisation par graphes multi-valués [Ziliaskopoulos and Wardell, 2000] est une représentation compacte dans laquelle les coûts sont définis sur chaque arc pour chaque mode de transport. Ce type de modélisation ne respecte pas la contrainte FIFO. En effet, si un arc peut être emprunté par un piéton ou par un autobus, il est évident que des situations non-FIFO se produiront. Cette dernière nécessite donc un développement d'algorithmes spécifiques de calcul d'itinéraires en surmontant la non satisfaction de la contrainte FIFO. La modélisation par graphe multi-couches [Lozano and Storchi, 2001] considère que chaque mode de transport constitue une partie du graphe de transport. Pour cela, les nœuds et les arcs du graphe sont étiquetés en fonction de la couche à laquelle ils appartiennent. Des couches spécifiques pour assurer le transfert entre les différents modes doivent être introduites pour compléter la modélisation. Une fois que les emplacements sont identifiés, les réseaux doivent être liés par l'insertion d'arcs de transfert entre les nœuds des différents réseaux. Une difficulté technique est l'identification de nœuds qui sont situés à proximité les uns des autres pour relier les différentes couches. Ce type de modélisation permet une extension directe des algorithmes classiques de calcul de plus court chemin, par exemple : l'algorithme de Martins [Gandibleux et al., 2006] qui est une extension de l'algorithme de Dijkstra dans un contexte multi-objectif.
La figure 2.13 représente un chemin multimodal projeté sur un graphe multi-couches.

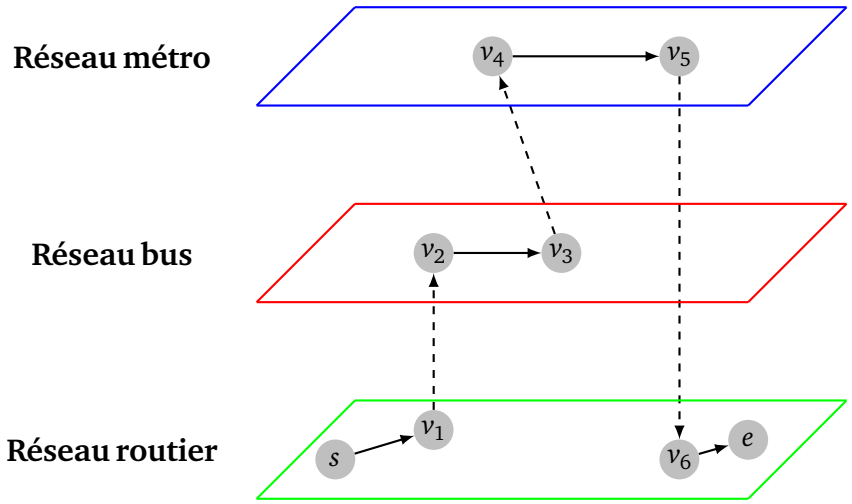


FIGURE 2.13 – Exemple de modélisation d'un réseau multimodal composé de trois couches. s et e correspondent respectivement à l'origine et la destination de la requête. Les arcs en pointillé représentent les arcs de transferts, tandis que les arcs en ligne continue correspondent aux chemins associés à chaque couche.

2.2 État de l'art sur la mobilité partagée

Comme expliqué auparavant, nous présentons dans cette section, une revue de littérature sur la mobilité partagée en décrivant les différentes formes existantes. Nous détaillons ensuite le problème de covoiturage classique.

2.2.1 Les problèmes de covoiturage et ses variantes

Les problèmes de covoiturage appartiennent aux problèmes de tournées de véhicules. Le problème classique de tournées de véhicules (VRP) fait parti des problèmes de recherche opérationnelle réputés NP-difficile [Lenstra and Kan, 1981]. Ce dernier consiste à visiter à l'aide de véhicules, différents lieux en minimisant le coût global des trajets. Dans le cas où la visite des différents lieux est faite par un seul véhicule, le problème se résumera au problème de voyageur de commerce (*Traveling Salesman Problem* (TSP)) connu pour être NP-difficile au sens fort. Une description historique complète de ce problème peut être trouvée dans [Hoffman et al., 2013]. En revanche, lorsqu'il s'agit de transporter des marchandises à partir des points de collecte vers des points de livraison, en utilisant plusieurs véhicules, ce problème correspondra au PDP (*Pickup & Delivery Problem ou Ramassage & Livraison*) [Savelsbergh and Sol, 1995]. L'ajout de points de collecte et de livraison implique des contraintes de priorité (la collecte avant la livraison) et de couplage (un même véhicule pour la collecte et la livraison). D'autre part, en transportant des personnes plutôt que des marchandises, le problème correspondra au *Dial-a-Ride Problem* (DARP) [Cordeau and Laporte, 2007]. En transportant des personnes, certaines contraintes supplémentaires sont introduites telles que l'obligation de respecter des délais particuliers, coûts de transport, etc.
Les *Transports A la Demande* (TAD) ou *Demand Responsive Transport* (DRT) [Basnal et al., 2015] au sens transport en commun routier, est un cas particulier de DARP où la qualité du service peut être mesurée en termes de confort, de convivialité, d'ergonomie, de formes des trajets par rapport aux

origines et aux destinations prises en compte en temps réel. Les TAD se distinguent des services de transports en commun car les véhicules n'empruntent pas d'itinéraires fixes et ne respectent pas un horaire précis, sauf pour satisfaire parfois un besoin particulier. A noter également que ces problèmes de planification de tournées peuvent se faire soit sur les nœuds ou sur les arcs. Dans les tournées sur nœuds (*node routing problems*), les tâches sont associées à des nœuds du réseau. Tandis que les tournées sur arcs (*arc routing problems*), le long des arcs (resp. arêtes) sont considérés comme des tâches. Comme par exemple, la collecte de déchets ménagers dont il faut ramasser les poubelles dans chaque rue. A travers les différents problèmes cités ci-dessus, il en existe une multitude de variantes, que l'on désigne généralement par des sigles pour les distinguer. Par exemple, la présence d'un seul ou plusieurs véhicules, de problèmes avec ou sans fenêtre de temps, statique ou dynamique, déterministe ou stochastique, etc.
Dans notre étude, nous nous intéressons uniquement aux problèmes de tournées sur nœuds plus exactement au problème de DARP qui permet de modéliser correctement la majorité de nos problèmes. Plus spécifiquement, nous considérons des sous problèmes polynomiaux du DARP général (dans un contexte de réseau de transport réel). Le DARP correspond à un problème de routage de véhicules où plusieurs clients précisent leur lieu de prise en charge et de destination, itinéraires des véhicules et les horaires devant être définis. Les clients précisent leur trajet habituellement accompagné de contraintes temporelles qui représentent le moment de prise en charge désiré et/ou le temps d'arrivée à destination. L'objectif habituel d'un DARP est de trouver pour chaque véhicule un ensemble de routes satisfaisant le plus grand nombre possible de demandes, tout en minimisant les inconvénients des clients par rapport aux contraintes de transport. Dans le problème de DARP, deux types de variantes peuvent être distinguées, à savoir, le DARP statique [Cordeau and Laporte, 2003] et le DARP dynamique [Deleplanque and Quilliot, 2015].
Le problème est dit statique si toutes les données sont connues sur l'horizon de temps où l'on doit calculer les tournées, c'est à dire de façon définitive préalablement au début du calcul des tournées de véhicules. Dans le cas dynamique, le système doit être capable d'intégrer des imprévus alors que des tournées ont déjà été planifiées et que leur exploitation a démarré. Ces imprévus peuvent être la panne ou le retard d'un véhicule, l'apparition ou la disparition d'une requête. Généralement, un premier algorithme calcule des tournées optimisées à partir des données connues. Un deuxième algorithme dynamique qui permet de prendre en compte les nouvelles données en respectant certains critères d'optimalité. Ceci se produit dans les applications « temps réel », d'où une contrainte supplémentaire de rapidité d'exécution pour l'algorithme dynamique.

La performance des algorithmes DARP dynamiques a été étudiée dans [Hyytiå et al., 2010], en particulier les critères de sélection d'itinéraires pour résoudre une prise en charge et son dépôt à grande échelle. Ils ont montré que l'attribution d'un nouveau client à chaque véhicule un par un (c'est à dire l'attribution immédiate) se traduit par une assez petite perte dans la qualité de la solution par rapport à une allocation optimale où les clients existants peuvent être réattribués à d'autres véhicules. En outre, ils ont étudié la performance d'un algorithme d'insertion où la séquence de clients est conservée pour chaque véhicule et les lieux de prise en charge et dépôt d'un nouveau client sont insérés dans cette séquence de façon optimale. Ils ont montré que la qualité de l'algorithme d'insertion de la solution diminue avec l'augmentation du nombre de clients par véhicule par rapport à un algorithme optimal d'énumération complète.

[Coslovich et al., 2006] ont proposé une technique d'insertion en deux phases pour les nouvelles demandes de trajets. La première phase est conçue pour fonctionner hors ligne, lorsque le véhicule est en mouvement et trouve un voisinage admissible à l'itinéraire actuel du véhicule. Le résultat est ensuite utilisé pour la deuxième phase qui est effectuée en ligne quand une nouvelle requête client

arrive. Cette phase tente d'insérer les lieux de prise en charge et de destination du nouveau client dans le voisinage admissible précédemment calculé. Il a été montré que cette approche peut produire des solutions en temps réel qui sont qualitativement proches d'une solution statique optimale.

[Häme, 2011] a étudié un algorithme d'insertion adaptatif pour le DARP avec des fenêtres de temps étroites qui pourraient également être utilisées dans le cas dynamique. Ces fenêtres rendent la plupart des nouveaux itinéraires potentiels inadmissible en essayant d'insérer un nouveau client dans un itinéraire existant. Grâce à cela, avec un paramètre limitant le nombre de tournées possibles considérés, leur algorithme d'insertion adaptatif réduit grandement l'espace de solutions et accélère la procédure d'insertion. De plus, l'algorithme comprend une adaptation dynamique du paramètre permettant d'élargir l'espace des solutions si aucune solution admissible n'est trouvée au premier abord.

2.2.2 Différentes formes de covoiturage

Dans ce qui suit, nous présentons quelques formes différentes de covoiturage. Le Tableau 2.1 présente des caractéristiques des trois formes les plus connues du covoiturage.

Forme de Covoiturage	Slugging	Taxi-partagé	Covoiturage classique
Emplacement de prise en charge	Origine fixe	Origine du passager	Origine du passager
Emplacement de dépose	Destination fixe	Destination du passager	Destination du passager
Flexibilité sur la route du conducteur	Non	Oui	Oui
Contrainte sur le temps de détour du conducteur	Non	Non	Oui

TABLE 2.1 – Caractéristiques de certains systèmes de partage.

Système de slugging

Dérivé du terme slug ou “fausse pièce de monnaie", il s'agit d'une pratique tout à fait originale du covoiturage utilisée dans certaines villes américaines. Les automobilistes s'arrêtent à un arrêt bien défini (signalé par une pancarte indiquant les points de slug) pour prendre un ou plusieurs passagers à leur bord et peuvent ainsi emprunter les voies rapides, en général fluides et réservées au covoiturage pour accéder plus rapidement au centre-ville. Le slugging est une initiative tout à fait unique visant à profiter des voies réservées aux véhicules à plus d'un passager (High Occupancy

Vehicle lane (HOV)) créées dans la majorité des grandes villes pour diminuer le volume du trafic le matin et aux heures de pointe. Ces voies encouragent les automobilistes à faire du covoiturage ou à utiliser les transports en commun. Ce sont dans ces circonstances que le slugging a vu le jour. Cette forme de covoiturage a d'abord débuté dans les années 1970 à Washington pour répondre à la hausse des prix créée par le choc pétrolier. Aujourd'hui, elle fait partie des transports courants de cette ville, si bien que les urbanistes espèrent l'encourager en construisant des zones qui lui seront spécialement destinées. Le "slugging" est un système d'autostop mais d'une façon plus organisée avec ses propres règles et son étiquette qui est tout à fait gratuite.
Le site *Slug-Lines* [11] donne tous les détails sur son fonctionnement et fournit aussi les cartes des emplacements pour Washington et quelques villes périphériques où les gens peuvent attendre les voitures. Parmi les règles à observer, il ne doit pas y avoir de conversations entre conducteurs et passagers sauf évidemment pour la communication de base. Aucun cadeau ou argent ne doit être échangé et enfin les conducteurs ne doivent pas s'arrêter pour faire une course en chemin.

Les conducteurs maintiennent leurs itinéraires initiaux et ne font pas de détour pour la prise en charge ou de dépose des passagers. Ainsi, le passager se dirige vers le point de départ du conducteur (arrêt spécifique), monte à bord à l'heure de départ. Il sera déposé à la destination du conducteur, puis enfin le passager rejoint sa propre destination. La gratuité de ce service s'applique en raison de l'avantage mutuel : les conducteurs peuvent utiliser les voies réservées et se rendre plus vite à leur destination alors que pour les passagers, le slugging est un moyen de transport gratuit et représente une grande économie de temps. Au final, le slugging a un impact important sur la réduction du trafic et donc de la pollution urbaine. Chaque partie a besoin de l'autre pour exister.
Le problème principal du *système de slugging* est l'inflexibilité des itinéraires des conducteurs, ce qui ne permet pas d'avoir un transport porte-à-porte.
L'étude [Ma and Wolfson, 2013] formalise ce problème ainsi que sa généralisation. Les auteurs se positionnent dans un contexte où les conducteurs et les passagers enregistrent leurs trajets à l'avance, ils forment ainsi un plan de regroupement des passagers en affectant un conducteur pour chaque groupe. La seule obligation attachée au passager est son déplacement à pied depuis son point de départ jusqu'à l'origine du conducteur (emplacement spécifique) et depuis la destination du conducteur (emplacement spécifique) jusqu'à sa propre destination. Leur approche est basée sur une détermination d'un sous-graphe maximal à partir d'un graphe initial constituant tous les trajets des passagers et conducteurs. La construction d'un sous-graphe maximal est déterminée de façon que les trajets des passagers soient émergés au mieux dans les différents itinéraires des conducteurs sous quelques contraintes. Ils proposent ainsi un algorithme quadratique pour le résoudre de manière optimale. Pour les variantes du problème de slugging qui sont contraints par la capacité du véhicule et le temps de trajet, les auteurs prouvent la NP-complétude du problème et proposent également quelques heuristiques de résolution. Ils développent également des heuristiques pour une version dynamique du problème en les évaluant sur des données réelles. D'autres travaux sur le problème de slugging sont présentés dans [Mote and Whitestone, 2011] et [Burris and Winn, 2006].

Système de taxi-partagé

Il désigne l'utilisation de taxis-cabines ordinaires par plus d'un passager à bord servant des voyages multiples dans le même trajet d'un taxi. Contrairement aux taxis-cabines, le coût d'un trajet dépend du nombre de passagers dans le taxi. Ce qui incite les passagers à une meilleure utilisation des taxis et qui évite également la sous-utilisation de sièges pendant les trajets.

11. http ://www.slug-lines.com/

Ce problème consiste à attribuer des véhicules de taxi pour satisfaire au mieux les requêtes des passagers qui sont répartis sur différents endroits dans une zone donnée. Plusieurs contraintes peuvent être considérées comme la capacité du véhicule, les fenêtres temporelles pour les passagers et les taxis. Plus spécifiquement, ce problème consiste à attribuer au sein de ces fenêtres de temps, l'instant pour chaque événement de prise en charge et de dépose. Contrairement au problème de slugging, la route des conducteurs (c.a.d chauffeurs de taxi) change pour servir les passagers en les prenant en charge depuis leur départ jusqu'à leur destination.

Les auteurs dans [Ma et al., 2013] propose un service de taxis pratique dans lequel une organisation exploite un service dynamique de taxi-partagé.
Dans leur système, les chauffeurs de taxis peuvent déterminer de manière indépendante quand ils rejoignent et quittent le service. Ainsi, les passagers soumettent des requêtes de trajets en temps réel via un appareil mobile, par exemple un Smart-phone. Chaque requête indique l'origine et la destination du trajet ainsi que les fenêtres de temps pendant lesquelles le passager veut être pris en charge et déposé. Une fois qu'une nouvelle requête est reçue, le système détermine un taxi le plus approprié qui est en mesure de satisfaire à la fois la nouvelle requête et les trajets des passagers à bord de ce taxi. Les mises à jour des horaires et itinéraires seront ensuite données pour le conducteur et les passagers du taxi correspondant.

Dans [Santos and Xavier, 2015], les auteurs étudient le problème de taxi-partagé combiné avec le covoiturage. Les conducteurs du covoiturage précisent leur lieu de départ, leur destination, l'heure de départ et le retard maximum toléré par ce dernier. Quant aux conducteurs de taxis, ces derniers indiquent leurs emplacements actuels ainsi que le temps de début et de fin du service. Les conducteurs doivent également fixer le prix par kilomètre ainsi que la capacité maximum de leur véhicule. Chaque passager a un coût maximal qu'il est prêt à payer pour son trajet. Leur approche de résolution est basée sur une discrétisation d'un problème dynamique en sous problèmes statiques. Plus exactement, pour résoudre ce problème dynamique, la journée est divisée en périodes de temps. Les requêtes sont mises en "*buffer*" pendant quelque temps, puis une version statique du problème est construite et résolue en utilisant la métaheuristique "*Greedy Randomized Adaptive Search Procedure*" (GRASP). Plus précisément, pour chaque période, une instance d'un problème statique est résolue par la métaheuristique GRASP. Un point important pour résoudre un problème dynamique est de calculer les plus courts chemins entre tous les points de départ et de destination d'une manière très efficace. Pour y remédier, les auteurs supposent que tous les points se trouvent dans un graphe déjà pré-traité représentant la carte d'une ville. Ils utilisent ensuite la "*Contraction Hiérarchique*" [Geisberger et al., 2012] pour accélérer le calcul des plus courts chemins entre les différentes paires. Ceci résout seulement une part du problème dans le sens où la prise en compte du trafic ainsi que les informations en temps réel sur les imprévus ne peuvent pas être gérées par l'approche "*Contraction Hiérarchique*". D'autres travaux sur le problème de taxi-partagé sont traités dans [d'Orey et al., 2012] et [Tao, 2007].

Système de covoiturage classique

Le principe est assez similaire au *Taxi-partagé*. Les principales différences sont :

(i) le covoiturage classique est basé sur des voitures personnelles dans lesquelles les passagers partagent leur trajet avec le conducteur, tandis que dans les taxis, la présence d'un conducteur de taxi agréé est obligatoire.

(ii) le taxi-partagé nécessite un mécanisme de tarification approprié afin d'inciter au mieux les chauffeurs de taxis. Cette tarification est généralement plus chère qu'un système de covoiturage.

Plusieurs études se sont focalisées ces dernières années sur les systèmes de covoiturage à la fois dans le milieu académique et industriel [Agatz et al., 2011] et [Furuhata et al., 2013]. Le principe du covoiturage classique est le suivant : un utilisateur indiquant son statut (c'est-à-dire conducteur ou passager), son point de départ et de destination, génère une requête par le biais de son Smartphone. La requête est envoyée à un serveur central. Les conducteurs (resp. passagers) potentiels sont ensuite sollicités, ils pourront accepter ou rejeter la requête. Si plusieurs conducteurs (resp. passagers) acceptent la demande, le conducteur (resp. passager) peut choisir selon ses préférences secondaires. En plus du statut de l'utilisateur, d'autres informations doivent être précisées. C'est-à-dire si l'utilisateur est identifié comme un conducteur, il doit préciser s'il est prêt à prendre un seul passager ou peut être disposé à en prendre plusieurs. D'autre part, s'il est identifié comme passager, il doit préciser s'il veut covoiturer avec un seul conducteur ou bien s'il est prêt à en prendre plusieurs. Ainsi, quatre variantes peuvent être distinguées : "un seul conducteur et un seul passager", "un seul conducteur et plusieurs passagers", "plusieurs conducteurs et un seul passager", et enfin "plusieurs conducteurs et plusieurs-passagers".
Le Tableau 2.2 montre les différents cas qui peuvent survenir lorsqu'un utilisateur entre dans un système de covoiturage.

	Un passager	Plusieurs passagers
Un conducteur	Appariement simple entre conducteur/passager	⋆ Un seul emplacement de prise en charge et de dépose ⋆ Plusieurs emplacements de prise en charge et de dépose
Plusieurs conducteurs	Passager est transféré entre plusieurs conducteurs	⋆ Routage des conducteurs ⋆ Routage des passagers

TABLE 2.2 – Variantes du système classique de covoiturage

Un-conducteur versus Un-passager.
Dans cette variante, chaque conducteur partage son trajet avec au plus un passager. Ainsi, pour un passager donné (resp. conducteur), le système détermine le meilleur conducteur (resp. passager) pour partager son trajet. Ce dernier sera choisi parmi un ensemble de conducteurs (resp. passagers) selon le critère souhaité. Les auteurs [Geisberger et al., 2010] considèrent le problème un seul passager et un seul conducteur. Leur objectif est de sélectionner le conducteur qui peut satisfaire la requête d'un passager avec le plus petit détour possible. Dans [Agatz et al., 2011], les auteurs étudient ce problème dans deux scénarios différents : le premier scénario où les utilisateurs fixent leur rôle à l'avance ; dans le second scénario, chaque utilisateur est flexible pour servir comme passager ou conducteur.
Pour le premier scénario, les auteurs modélisent le problème comme un problème de couplage maximum dans un graphe biparti avec quelques contraintes supplémentaires. Le premier ensemble est composé de conducteurs tandis que le second est composé de passagers. Deux nœuds des deux ensembles sont reliés si et seulement si ces derniers respectent les contraintes d'appariement. Ils

résolvent ce problème par la programmation linéaire en transformant ce dernier en problème de flot maximum après l'ajout d'un nœud source et d'un nœud cible au graphe biparti. Pour le second scénario, l'objectif n'est plus seulement l'affectation des passagers aux conducteurs, mais également l'attribution de rôles pour chaque participant pour construire des appariements optimaux entre les conducteurs et les passagers. La modélisation par un graphe biparti n'est plus possible dans le second scénario. Pour y remédier, les auteurs considèrent un graphe plus général avec quelques adaptations pour le problème de couplage maximum. Ils explorent ainsi différentes approches (programmation linéaire en nombres entiers et une approche gloutonne) pour l'appariement des conducteurs et des passagers en temps réel et ils ont étudié également l'impact des différentes caractéristiques du système. Leur étude montre que le succès d'un système de covoiturage dépend fortement de la densité des participants, à savoir le nombre de participants au km^2, d'une densité minimum de participants nécessaire pour assurer un système stable (dans lequel les participants ne quittent pas le système même si à plusieurs reprises ils ne parviennent pas à trouver d'appariement).

Un-conducteur versus Plusieurs-passagers.

Pour cette variante, deux cas peuvent se présenter :

(i) Premier cas, le conducteur est prêt pour prendre en charge chaque passager à son point d'origine et le déposer à sa destination. Le système doit déterminer une liste de passagers et un ordre de prise en charge et de dépose. Les auteurs [Baldacci et al., 2004] proposent une méthode exacte ainsi qu'une méthode approchée pour résoudre ce problème. Leurs approches sont basées sur la programmation en nombres entiers. Leur méthode exacte est basée sur une procédure de "bounding" qui combine trois bornes inférieures provenant de différentes relaxations du problème principal. Une borne supérieure valide est obtenue par une approche heuristique qui transforme la solution d'une borne inférieure lagrangienne à une solution réalisable. Dans [Wolfler Calvo et al., 2004], les auteurs développent une heuristique basée sur une recherche locale. Dans [Herbawi and Weber, 2012b], les auteurs modélisent le problème d'appariement avec fenêtres de temps comme un problème d'optimisation pour lequel ils proposent un algorithme génétique pour le résoudre. Leur fonction objectif est une fonction mono-objectif qui est une somme pondérée de plusieurs critères. Cette dernière consiste à minimiser le temps total du trajet, les distances des trajets des conducteurs, le temps de trajet total des passagers et enfin la maximisation du nombre d'appariement.

(ii) Il s'agit du cas où le conducteur souhaite prendre en charge plusieurs passagers dans un même emplacement et les déposer au même endroit. Le système détermine une liste de passagers, le meilleur emplacement de prise en charge parmi leurs emplacements de départ ainsi qu'un meilleur emplacement de dépose parmi leurs destinations. A notre connaissance, la seule étude similaire à cette variante a été introduite dans [Stiglic et al., 2015]. Ils proposent une formulation basée sur une extension du modèle proposé dans [Agatz et al., 2011]. Dans leur modèle, chaque conducteur à une possibilité d'avoir au plus un seul lieu de prise en charge et un seul lieu de dépose pour l'ensemble de ses passagers. En d'autres termes, les passagers seront pris en charge en même temps dans un seul emplacement de prise en charge et ils seront également déposés en même temps dans un autre lieu de dépose. Les emplacements de prise en charge et de dépose sont fixés à des lieux stratégiques tels que des arrêts de bus, des stations essence, etc. Les auteurs se basent sur de fortes hypothèses, à savoir : les emplacements des passagers et des conducteurs dans un plan euclidien, les distances sont euclidiennes et qu'un trajet (à pied ou en voiture) se produit à une vitesse constante. La plupart de ces hypothèses sont difficilement extensibles de manière à couvrir des paramètres plus réalistes.
Le principal objectif de leur étude est plutôt axé sur l'intérêt de considérer des points de rencontre dans un service de covoiturage.

Plusieurs-conducteurs versus Un-passager.
Cette variante dans laquelle un passager est transféré entre plusieurs conducteurs est étudiée dans [Herbawi and Weber, 2011, Drews and Luxen, 2013]. Ce problème particulier du covoiturage est souvent appelé "covoiturage multi-sauts".
Dans [Herbawi and Weber, 2011], le problème est modélisé comme un problème de plus court chemin multi-objectifs, sur un graphe *Time-expanded* [Pyrga et al., 2008], dans lequel les conducteurs ne dévient pas de leurs itinéraires initiaux et de leurs horaires. Leur objectif est de trouver une route qui minimise les coûts totaux de déplacement et le nombre de transferts.
Les auteurs dans [Drews and Luxen, 2013] étend ce travail en permettant également des détours raisonnables pour les conducteurs. Un algorithme efficace qui permet de générer un plus court chemin satisfaisant les exigences de "multi-sauts en covoiturage" a été présenté. Les auteurs fixent en amont un nombre élevé de stations dans un réseau routier où les passagers peuvent passer d'un véhicule à un autre. Leur modèle a été également résolu par une exploitation des graphes de type *Time-expanded* qui sont utilisés pour le routage dans les réseaux de transports. Chaque nouvelle offre est insérée dans le graphe *Time-expanded*. L'insertion de cette dernière nécessite le parcours des stations de transferts en vérifiant la faisabilité de quelques contraintes, à savoir le délai maximum et le temps de détour maximal. Le temps d'insertion d'une offre est très rapide dû à la méthode d'accélération utilisée (*Contraction hiérarchique*). Finalement, le graphe *Time-expanded* induit est composé de différents trajets de conducteurs en passant par les différentes stations relais possibles à des intervalles de temps bien définis. Deux arcs consécutifs dans ce graphe *Time-expanded* assurent la synchronisation en temps entre les deux trajets associés à ces derniers. Ainsi, lorsqu'une demande de covoiturage arrive dans le système, la position de départ et la destination de la demande seront projetés dans le graphe *Time-expanded*. Ensuite, il reste à calculer le plus court chemin depuis son point de départ jusqu'à sa destination en respectant les contraintes sur la somme des délais ainsi que le détour totale du passager.

Plusieurs-conducteurs versus Plusieurs-passagers.
Ce problème consiste à déterminer un acheminement simultané pour les passagers ainsi que pour les conducteurs. Plus précisément, un conducteur peut prendre plusieurs passagers, ces derniers peuvent être parallèlement transférés entre plusieurs conducteurs. L'étude dans [Herbawi and Weber, 2012a] propose un algorithme génétique pour résoudre ce problème. Leur modèle est une extension de l'approche proposée dans [Herbawi and Weber, 2012b] en relaxant la contrainte sur le nombre de transferts d'un passager à deux conducteurs. Plus spécifiquement, il réajuste l'algorithme génétique proposé en changeant la formulation du chromosome pour prendre en compte la possibilité d'avoir un point de transfert dans une tournée créée. Ce changement engendre des modifications dans la génération de la population initiale, mutation et croisement dans le sens où deux tournées peuvent être corrélées entre elles en ayant un même point de transfert (d'où la nécessité de rajouter une contrainte sur la synchronisation temporelle entre les deux conducteurs sur ce lieu de transfert). Leur modèle est sensible au nombre de transferts en générant un nombre élevé de combinaisons possibles entre les différentes tournées créées.

2.3 Conclusion du chapitre

Dans ce chapitre, les différentes notions qui vont être utilisées dans la suite ont été présentées. Pour ce faire, nous avons commencé par présenter les notions de graphes ainsi que leurs propriétés. Ensuite, nous avons présenté les problèmes de cheminement sur les graphes statiques, les algorithmes de plus court chemin ainsi que quelques techniques d'accélération.
Dans un second temps, nous avons présenté différents types de modélisations pour les graphes non statiques (réseaux routiers), dynamiques (réseaux de transports en commun) et un modèle qui permet de modéliser un réseau de transport multimodal en combinant les différents modes de transport. Ensuite, nous avons proposé une revue de littérature sur les différentes mobilités partagées existantes.

3

Covoiturage dynamique avec des emplacements intermédiaires de rencontre et de séparation

Sommaire

Dans ce chapitre, nous considérons des systèmes de covoiturage avec points intermédiaires de rencontre et de séparation. Plus précisément, ces systèmes de covoiturage offrent aux passagers et aux conducteurs la possibilité de se rencontrer et de se séparer dans des emplacements intermédiaires autres que leur position de départ et d'arrivée. L'idée principale de nos approches consiste à étiqueter certains nœuds d'une carte géographique avec des informations sur les conducteurs, ce que l'on appelle des *buckets*. Basé sur ces informations, lorsqu'un passager entre dans le système, nous

déterminons un meilleur conducteur ainsi qu'un meilleur emplacement de prise en charge et de dépose qui minimisent le coût du trajet total du passager et du conducteur sous certaines contraintes. Une méthode exacte et plusieurs heuristiques sont proposées afin d'identifier un meilleur conducteur, de meilleurs emplacements de prise en charge et de dépose ainsi que le taux de partage des coûts. Ensuite, nous procédons à une évaluation comparative en utilisant un vrai réseau routier de la région Lorraine en France et un vrai jeu de données fourni par une entreprise de covoiturage Covivo [12].
Les résultats obtenus montrent l'efficacité de notre système par rapport au covoiturage classique en termes de taux d'appariement et d'économies de coûts tout en restant dans un délai d'exécution raisonnable.

3.1 Introduction

Nous traitons dans ce chapitre le problème où les passagers ont la possibilité de se déplacer vers des lieux intermédiaires de prise en charge et de dépose autre que leurs lieux de départ et de destination. Cela évite d'obliger le conducteur d'aller chercher le passager depuis son point départ pour le transporter jusqu'à sa destination. La prise en compte de ces emplacements intermédiaires permet de réduire considérablement le détour pour le conducteur ainsi que le coût total du trajet (voir la figure 3.1). Deux études récentes qui proposent des emplacements intermédiaires de prise en charge et de dépose sont présentées dans [Bit-Monnot et al., 2013] et [Stiglic et al., 2015]. L'approche proposée dans [Bit-Monnot et al., 2013] sera présentée dans le chapitre 6. La différence principale réside dans l'introduction d'une contrainte sur le détour du conducteur/passager et l'introduction d'une fonction de cout au lieu d'une fonction calculant le temps de trajet pour évaluer la performance d'un covoiturage. Dans [Stiglic et al., 2015], les passagers sont pris en charge en même temps sur un seul emplacement de départ et ils sont également déposés sur un même lieu de dépose. Leur objectif est de maximiser le nombre d'appariements dans le système. Cependant, les hypothèses fixées par les auteurs réduisent considérablement la difficulté du problème car aucun calcul de plus court chemin ne s'effectue lors d'une arrivée d'une requête. Le principal objectif de leur étude est de montrer l'intérêt de considérer des points de rencontre et de séparation pour le service de covoiturage.
Contrairement à leur approche, nous utilisons des durées exactes qui seront calculées dynamiquement lors d'une demande de covoiturage. Le passager sera également en mesure de choisir son mode de transport pour atteindre le lieu de prise en charge depuis son point de départ ainsi qu'un second mode de transport pour atteindre sa destination depuis le lieu de dépose. Dans ce chapitre, les modes de transport possibles pour le passager concernent uniquement les transports individuels, à savoir : le vélo privé, la voiture personnelle, la marche à pied. A l'inverse des transports en commun, leurs disponibilités ne dépendent pas du temps.

12. www.covivo.eu

(a) Avant l'appariement

(b) Après l'appariement

FIGURE 3.1 – La figure (a) représente le plus court chemin du passager (de couleur verte) et du conducteur (de couleur rouge) avant l'appariement. La figure (b) représente les nouveaux chemins du conducteur et du passager après l'appariement. Les deux nouveaux emplacements intermédiaires de prise en charge et de dépose sont représentés par deux flêches noires, tandis que le chemin commun est représenté en bleu (le chemin se trouvant entre les deux emplacements intermédiaires). L'approche classique de covoiturage ne détecte pas cet appariement dû au long détour du conducteur pour transporter le passager directement depuis sa position de départ à sa destination.

Le reste de ce chapitre est constitué comme suit : la section 3.2 décrit le modèle de covoiturage dynamique avec des emplacements intermédiaires de rencontre et de séparation. La section 3.3 présente les différentes approches de résolution du problème présenté dans la section 3.2. Dans la section 3.4 nous traiterons le problème de sélection du meilleur conducteur lorsqu'une demande de covoiturage entre dans le système. En effet, les approches développées dans la section 3.3 correspondent au cas où une offre et une demande de covoiturage sont déjà fixées à l'avance. Une analyse comparative entre notre modèle de covoiturage avec emplacements intermédiaires et le modèle classique sur différents réseaux routiers et différentes données (réelles et simulées) est présentée dans la section 3.5. Enfin, nos conclusions du chapitre sont présentées dans la section 3.6.

3.2 Description et notations

Un réseau routier est généralement représenté par un graphe orienté $G = (V,A)$, où V est l'ensemble de nœuds et A l'ensemble d'arcs. L'ensemble de nœuds V correspond à des intersections de routes et l'ensemble d'arcs A correspond à des liens possibles pour passer d'un nœud à un autre. Dans notre graphe, pour chaque arc $(i,j) \in A$ nous associons deux poids positifs $c(i,j)$ et $\tau(i,j)$, qui représentent respectivement le coût et le temps pour passer du nœud i au nœud j. Un chemin μ dans G est défini par un ensemble ordonné de nœuds $\mu =< u,\ldots,v >$ avec $\mu \subseteq V$ et $l = |\mu|$ la longueur du chemin μ. Pour chaque couple de nœuds consécutifs i et j du chemin μ, il existe un arc (i,j) reliant ces deux nœuds. Le coût $c(\mu)$ du chemin μ est la somme des coûts des arcs dans μ. Un plus court chemin entre une source u et une destination v est un chemin avec un coût minimal parmi tous les chemins de u à v. Dans ce qui suit, un plus court chemin entre u et v sera représenté par $u \rightarrow v$. En revanche, les emplacements de départ et d'arrivée du conducteur et du passager seront respectivement notés par (s,e) et (s',e'). Une requête d'un utilisateur k peut correspondre soit à une offre de covoiturage $k = o$ ou bien à une demande de covoiturage $k = d$. Ainsi, une offre et une demande de covoiturage sont représentées respectivement par $o = (s,e,[t_o^{\min}, t_o^{\max}],\Delta_o)$ et $d = (s',e',[t_d^{\min}, t_d^{\max}],\Delta_d)$, où $[t_o^{\min}, t_o^{\max}]$ et $[t_d^{\min}, t_d^{\max}]$ sont les fenêtres de temps respectives du conducteur et du passager. Le temps de détour pour le conducteur et le passager sont représentés respectivement par Δ_o et Δ_d.

Généralement, la structure de la tarification se distingue en trois parties différentes :

- La tarification unique (même tarif sans prendre en compte la distance parcourue sur une ligne particulière).
- La tarification échelonnée (les prix augmentent avec la distance parcourue sur une ligne particulière).
- La tarification par zones (les tarifs augmentent avec la distance parcourue suivant des seuils de tarifs et ils sont dépendants également du type de route comme le péage).

Dans notre étude, nous nous positionnons dans une tarification échelonnée dans laquelle les coûts sont calculés selon la distance parcourue et non selon des tarifs fixés.
Les auteurs dans [Geisberger et al., 2010] développent une approche basée sur une tarification échelonnée qui permet de sélectionner les meilleures offres avec les plus petits détours pour une demande de covoiturage donnée. Ainsi, une offre apparie parfaitement une demande de covoiturage si et seulement si le conducteur et le passager ont le même le lieu de départ et de destination. L'appariement dans lequel le conducteur fait un petit détour pour transporter un passager est dit "appariement raisonnable"

Definition. 3 (Appariement raisonnable dans un système de covoiturage classique) *Une offre de covoiturage* $o = (s,e,[t_o^{\min}, t_o^{\max}],\Delta_o)$ *et une demande de covoiturage* $d = (s',e',[t_d^{\min}, t_d^{\max}],\Delta_d)$

forment un appariement raisonnable si et seulement si il existe un chemin $\gamma_{od} = s \to s' \to e' \to e$ *dans le graphe G telle que* $c_o(\gamma_{od}) \leq (c_o(s \to e) + \varepsilon \cdot c_o(s' \to e'))$, *avec* $\varepsilon \geq 0$.

La valeur de ε peut être vue comme un taux de partage de coûts entre le passager et le conducteur sur le trajet commun $s' \to e'$. Autrement dit, le terme $\varepsilon \cdot c_o(s' \to e')$ est le paiement que le passager donne au conducteur sur le trajet commun. Dans cet arrangement, on incite le conducteur à prendre le passager à son emplacement de départ s' et le déposer à sa destination e'. En effet, si $c_o(\gamma_{od}) - c_o(s \to e) > \varepsilon \cdot c_o(s' \to e')$, le conducteur ne sera pas en mesure d'accepter la demande d. Un choix raisonnable pour ε est de ne pas excéder la valeur 1, sinon le coût encouru pour le passager sera plus élevé que son trajet direct. Lorsque ε=0, cela implique que le point de départ ainsi que la destination du passager se trouvent sur l'un des plus courts chemins du conducteur lorsqu'il voyage seul. Dans un tel cas, aucun détour n'est effectué par le conducteur et le coût du nouveau trajet reste inchangé. Cette approche est inflexible et les possibilités d'appariement des conducteurs et des passagers diminuent lorsque les itinéraires des passagers et des conducteurs sont légèrement différents. Cependant, afin d'obtenir plus de possibilités de covoiturage, le passager peut accepter deux emplacements intermédiaires : r_1 pour la prise en charge et r_2 pour la dépose. Plus spécifiquement, le passager se déplace par ses propres moyens au premier lieu intermédiaire r_1 avec un coût $c_d(s' \to r_1)$, là où il est pris en charge par le conducteur avant qu'il ne soit déposé au deuxième emplacement intermédiaire r_2 et enfin il rejoint sa destination finale de r_2 à e'. Dans ce cas, le passager et le conducteur partagent le coût du trajet commun $c_o(r_1 \to r_2)$. Si le coût du passager sur le trajet commun est de $\varepsilon \cdot c_o(r_1 \to r_2)$, alors son coût global est de $c_d(s' \to r_1) + \varepsilon \cdot c_o(r_1 \to r_2) + c_d(r_2 \to e')$. Selon la valeur de ε fixée, deux cas peuvent être distingués : une *approche a priori* et une *approche a posteriori*.

(i) ***Approche a priori*** dans laquelle l'utilisateur (conducteur/passager) fixe au préalable le taux de partage de coûts ε. Par exemple, le conducteur veut partager équitablement le coût du chemin commun avec le passager, c'est à dire $\varepsilon = 0.5$.

(ii) ***Approche a posteriori*** dans laquelle l'utilisateur (conducteur/passager) ne fixe pas à l'avance le taux de partage des coûts ε. En revanche, le système détermine la valeur la plus attractive de ε qui garantie un meilleur appariement entre le passager et le conducteur.

Dans les deux approches, la fonction objectif reste la même. La seule différence réside dans les contraintes d'appariement qui nécessitent différentes approches de résolution.

3.2.1 Objectif du système de covoiturage avec des points de rencontre et de séparation

Nous utiliserons le terme *trajet-total* pour décrire la concaténation des trajets $s \to r_1$, $s' \to r_1$, $r_1 \to r_2$, $r_2 \to e$ et $r_2 \to e'$. c'est à dire, $(s, s', r_1, r_2, e, e') = (s \to r_1 \oplus s' \to r_1 \oplus r_1 \to r_2 \oplus r_2 \to e \oplus r_2 \to e')$. Un plus court *trajet-total* entre les deux sources s, s' et les deux destinations e, e' est un *trajet-total* avec un coût minimal $GlobalPath(s, s', r_1, r_2, e, e')$ parmi tous les *trajet-totaux* de s, s' à e, e', où,

$$GlobalPath(s, s', r_1, r_2, e, e') = c_o(s \to r_1) + c_d(s' \to r_1) + c_o(r_1 \to r_2) + c_o(r_2 \to e) + c_d(r_2 \to e') \quad (3.1)$$

Ainsi, l'objectif de ce système est de déterminer deux meilleurs emplacements intermédiaires r_1 et r_2 qui minimisent le plus court *trajet-total* tel que l'offre o et la demande d forment un appariement raisonnable.

La Figure (3.2) représente un exemple d'appariement entre un conducteur et un passager en intégrant la notion de deux emplacements intermédiaires.

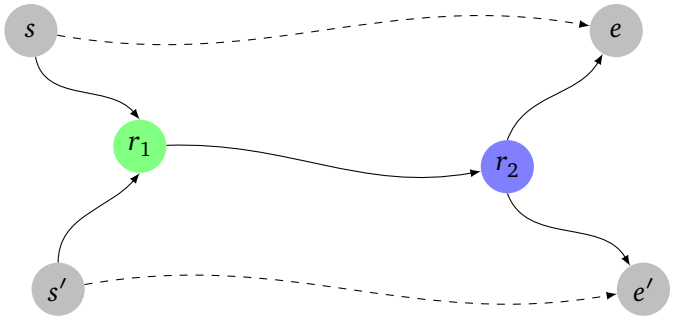


FIGURE 3.2 – Les lignes continues représentent le chemin du conducteur et du passager formant un appariement avec deux emplacements intermédiaires r_1 et r_2. En revanche, les deux lignes en pointillé représentent les plus courts chemins du conducteur et du passager voyageant séparément.

3.2.2 Contraintes d'appariement

Dans un système de covoiturage, un appariement entre conducteur et passager peut se réaliser seulement si certaines contraintes d'appariement sont satisfaites. Dans nos approches, nous en considérons deux importantes : la contrainte de timing ainsi que la contrainte de coût et du temps du trajet .

Contrainte de timing

La contrainte de timing est l'une des plus importantes dans un système de covoiturage dynamique. Ainsi, le choix du nouvel emplacement de prise en charge où le conducteur et le passager peuvent se rencontrer dépend de leur fenêtre de temps de départ. Cet emplacement est admissible pour une prise en charge si et seulement si il existe une synchronisation temporelle entre le conducteur et le passager qui est définit comme suit ;

Definition. 4 (Synchronisation temporelle)
Une demande d et une offre o forment une synchronisation temporelle à l'emplacement intermédiaire v si et seulement si $\beta \geq 0$, où,

$$\beta = \min \begin{cases} t_o^{\max} + \tau_o(s \to v) - (t_d^{\min} + \tau_d(s' \to v)) & \text{(3.2a)} \\ t_d^{\max} + \tau_d(s' \to v) - (t_o^{\min} + \tau_o(s \to v)) & \text{(3.2b)} \end{cases}$$

L'équation (3.2a) garantit au passager d quittant son emplacement de départ s' à l'instant $t_d^{\min}$, une arrivée pas plus tard que l'heure d'arrivée maximale du conducteur à l'emplacement intermédiaire v. Le même raisonnement est appliqué pour le conducteur dans l'équation (3.2b). Ainsi, lorsque β est positif, la propagation des deux fenêtres de temps du conducteur et du passager à l'emplacement v est non nul. Autrement dit, les deux fenêtres de temps d'arrivées du conducteur et du passager à l'emplacement v auront une intersection non vide. Ainsi, l'heure au plus tôt à laquelle ils peuvent se rencontre dans cette emplacement est $\max\left\{t_d^{\min} + \tau_d(s' \to v), t_o^{\min} + \tau_o(s \to v)\right\}$.

Contraintes de coût et du temps de trajet

L'économie des coûts est l'une des principales motivations pour que les usagers utilisent le service de covoiturage, auquel s'ajoute la protection de l'environnement. Cependant, chaque utilisateur

(conducteur/passager) doit également fixer la durée supplémentaire tolérée en plus de sa durée directe.

Pour garantir ces contraintes, nous définissons la notion d'un *appariement raisonnable* entre un conducteur et un passager pour l'approche *a priori* et *a posteriori*.
La définition 5 correspond à un *appariement raisonnable* dans une approche *a priori*, tandis que la définition 6 correspond à l'approche *a posteriori*.

Definition. 5 (Appariement raisonnable dans un système de covoiturage a priori avec points de rencontre) *Soit $\varepsilon \geq 0$ (fixé par le conducteur), une offre $o = (s, e, [t_o^{\min}, t_o^{\max}], \Delta_o)$ et une demande $d = (s', e', [t_d^{\min}, t_d^{\max}], \Delta_d)$ forment un appariement raisonnable si et seulement si il existe deux emplacements intermédiaires r_1 et r_2 ($r_2 \neq r_1$) tel que r_1 forme une synchronisation temporelle entre le conducteur et le passager et que les contraintes suivantes soient satisfaites.*

$$c_o(s \to r_1) + c_o(r_1 \to r_2) + c_o(r_2 \to e) - c_o(s \to e) \leq \varepsilon \cdot c_o(r_1 \to r_2) \tag{3.3}$$
$$c_d(s' \to r_1) + c_o(r_1 \to r_2) + c_d(r_2 \to e') - c_d(s' \to e') \leq (1-\varepsilon) \cdot c_o(r_1 \to r_2) \tag{3.4}$$
$$\tau_o(s \to r_1) + \tau_o(r_1 \to r_2) + \tau_o(r_2 \to e) \leq \tau_o(s \to e) + \Delta_o \tag{3.5}$$
$$\tau_d(s' \to r_1) + \tau_o(r_1 \to r_2) + \tau_d(r_2 \to e') \leq \tau_d(s' \to e') + \Delta_d \tag{3.6}$$

La contrainte (3.3) assure pour le passager des économies de coût sur son nouveau trajet en covoiturage $r_1 \to r_2$ tout en incluant ses coûts de rabattement depuis s' vers r_1 et de r_2 vers e'. De même, la contrainte (3.4) assure au conducteur des économies de coût sur son nouveau trajet en vérifiant si le coût du passager sur le trajet commun couvre au moins le coût du détour effectué. Le terme $(\tau_o(s \to e) + \Delta_o)$ dans (3.5) (resp. $\tau_d(s' \to e') + \Delta_d$ dans (3.6)) permet de limiter le temps que le conducteur (resp. passager) peut passer dans son trajet. Le temps de service pour la prise en charge et de dépose des passagers n'est pas significatif et peut être considéré comme instantané.

Definition. 6 (Appariement raisonnable dans un système de covoiturage a posteriori avec points de rencontre) *Une offre $o = (s, e, [t_o^{\min}, t_o^{\max}], \Delta_o)$ et une demande $d = (s', e', [t_d^{\min}, t_d^{\max}], \Delta_d)$ forment un appariement raisonnable si et seulement si il existe deux locations intermédiaires r_1 et r_2 ($r_2 \neq r_1$) tel que r_1 forme une synchronisation temporelle entre le conducteur et le passager, les contraintes (3.5) et (3.6) sont satisfaites, et que*

$$c_o(s \to e) + c_d(s' \to e') - \big(c_o(s \to r_1) + c_d(s' \to r_1) + c_o(r_1 \to r_2) + c_o(r_2 \to e) + c_d(r_2 \to e')\big) \geq 0 \tag{3.7}$$

Si les deux contraintes (3.3) et (3.4) sont satisfaites, alors la contrainte (3.7) est réalisable, le sens inverse n'est pas toujours vrai. Plus exactement, l'inégalité (3.7) représente les coûts économisés en appariant l'offre o avec la demande d par rapport aux coûts respectifs si chacun voyageait seul de son côté. Cette dernière est définie comme la différence entre le coût du trajet du conducteur et du passager lorsque chacun d'entre eux se déplace seul sans covoiturage, et le coût du trajet total du passager et du conducteur avec un trajet en commun entre r_1 et r_2.
Dans le lemme suivant, nous montrons que si la réduction des coûts générés est positive, alors il est possible de répartir les coûts économisés entre le conducteur et le passager de sorte que chacun d'entre eux ait un avantage.

Lemme. 1 *Si une offre $o = (s, e, [t_o^{\min}, t_o^{\max}], \Delta_o)$ et une demande $d = (s', e', [t_d^{\min}, t_d^{\max}], \Delta_d)$ forment un appariement raisonnable, alors il existe un taux de partage de coûts $\varepsilon \in [0, 1]$, tel que les gains du passager et du conducteur soient positifs.*

Preuve. 1 *Supposons qu'une offre o et une demande d forment un appariement raisonnable.*
D'après la définition (6), il existe deux emplacements intermédiaires r_1 et r_2, de sorte que le passager et le conducteur forment une synchronisation temporelle en r_1 et que la contrainte suivante soit satisfaite :

$$c_o(s \to e) + c_d(s' \to e') - \big(c_o(s \to r_1) + c_d(s' \to r_1) + c_o(r_1 \to r_2) + c_o(r_2 \to e) + c_d(r_2 \to e')\big) \geq 0$$

Supposons qu'il existe un taux de partage de coûts $\varepsilon \in [0,1]$ dans lequel le coût du trajet commun du passager et du conducteur est partagé selon ε, c'est à dire le passager récompense le conducteur avec un montant de $\varepsilon \cdot c_o(r_1 \to r_2)$.
D'une part, le conducteur accepte de partager son trajet avec le passager que si son coût total en utilisant le covoiturage est inférieur à son coût s'il voyageait seul, c'est à dire :

$$c_o(s \to r_1) + (1-\varepsilon) \cdot c_o(r_1 \to r_2) + c_o(r_2 \to e) \leq c_o(s \to e) \tag{3.8}$$

De (3.8) on obtient, $\varepsilon \geq \varepsilon_1$, où

$$\varepsilon_1 = \frac{c_o(s \to r_1) + c_o(r_1 \to r_2) + c_o(r_2 \to e) - c_o(s \to e)}{c_o(r_1 \to r_2)}$$

D'autre part, le passager accepte d'être pris en charge et déposé par le conducteur dans des emplacements intermédiaires que si son coût total en utilisant le covoiturage est inférieur à son coût s'il voyageait seul, c'est à dire :

$$c_d(s' \to r_1) + \varepsilon \cdot c_o(r_1 \to r_2) + c_d(r_2 \to e') \leq c_d(s' \to e') \tag{3.9}$$

De (3.9) on obtient, $\varepsilon \leq \varepsilon_2$, où

$$\varepsilon_2 = \frac{c_d(s' \to e') - c_d(s' \to r_1) - c_d(r_2 \to e')}{c_o(r_1 \to r_2)}$$

Cependant, il est évident que l'existence d'une solution réalisable est limitée par $\varepsilon_1 \leq \varepsilon \leq \varepsilon_2$.
Afin de satisfaire la contrainte (3.7), ε doit être compris dans l'intervalle $[\varepsilon_1, \varepsilon_2]$. Autrement dit, toute valeur de ε dans cette intervalle satisfait la contrainte d'appariement entre le conducteur et le passager. Une valeur raisonnable de ε serait la valeur moyenne de l'intervalle $[\varepsilon_1, \varepsilon_2]$, c'est à dire $\varepsilon = \frac{\varepsilon_1 + \varepsilon_2}{2}$. Cette valeur correspond au taux de partage équitable des coûts entre le conducteur et le passager sur le trajet commun dans le cas où ε n'est pas fixé au préalable par les utilisateurs.

3.3 Approches de résolution

3.3.1 Espace de recherche de lieux intermédiaires de prise en charge et de dépose

Les contraintes d'appariement définies dans la section 3.2.2 nous permettent de limiter considérablement l'espace de recherche de lieux potentiels de prise en charge et de dépose. Dans ce qui suit, nous proposons les caractéristiques associées à ces derniers.

Definition. 7 (Lieux potentiels de prise en charge et de dépose)
Soient $N^{\uparrow}(s)$, $N^{\uparrow}(s')$, $N^{\downarrow}(e)$ et $N^{\downarrow}(e')$ des ensembles des nœuds définis comme suit.

$$N^{\uparrow}(s) = \{v \in V \mid c_o(s \to v) + (1-\varepsilon) \cdot c_o(v \to e) \leq c_o(s \to e) \\ \wedge\ \tau_o(s \to v) + \tau_o(v \to e) \leq \tau_o(s \to e) + \Delta_o\}$$

$$N^{\downarrow}(e) = \{v \in V \mid (1-\varepsilon)\cdot c_o(s \to v) + c_o(v \to e) \leq c_o(s \to e) \\ \wedge\, \tau_o(s \to v) + \tau_o(v \to e) \leq \tau_o(s \to e) + \Delta_o\}$$

$$N^{\uparrow}(s') = \{v \in V \mid c_d(s' \to v) + \varepsilon \cdot \hat{d}(v \to e') \leq c_d(s' \to e') \\ \wedge\, \tau_d(s' \to v) + \hat{\tau}(v \to e') \leq \tau_d(s' \to e') + \Delta_d\}$$

$$N^{\downarrow}(e') = \{v \in V \mid \varepsilon \cdot \hat{d}(s' \to v) + c_d(v \to e') \leq c_d(s' \to e') \\ \wedge\, \hat{\tau}(s' \to v) + \tau_d(v \to e') \leq \tau_d(s' \to e') + \Delta_d\}$$

$N^{\uparrow}(s)$ représente l'ensemble de lieux potentiels de prise en charge pour le conducteur. Les deux contraintes qui définissent cet ensemble correspondent aux bornes inférieures des deux contraintes, à savoir : le temps et le coût du détour. En effet, si v est un lieu de prise en charge, la meilleure situation pour le conducteur serait de partager avec le passager la totalité du trajet restant jusqu'à sa destination e, c'est à dire le coût $c_o(v \to e)$ tout en respectant la contrainte du temps $\tau_o(s \to v) + \tau_o(v \to e) \leq \tau_o(s \to e) + \Delta_o$. Ainsi, si un tel nœud v ne respecte pas l'une des deux bornes supérieures sur la contrainte de coût ou de temps de déplacement, alors v ne pourra jamais correspondre à un lieu de prise en charge. L'ensemble $N^{\downarrow}(e)$ contient les lieux potentiels de dépose. Le même raisonnement s'applique à $N^{\downarrow}(e)$. Plus précisément, en prenant v comme emplacement de dépose intermédiaire, la meilleure situation qui peut se produire pour le conducteur dans ce cas est de partager avec le passager le coût du trajet depuis son origine s jusqu'au lieu de dépose $c_o(s \to v)$. D'autre part, les deux ensembles $N^{\uparrow}(s')$ et $N^{\downarrow}(e')$ représentent respectivement les lieux potentiels de prise en charge et de dépose pour le passager.

Concernant la contrainte du temps de trajet dans les deux ensembles $N^{\uparrow}(s')$ et $N^{\downarrow}(e')$, nous utilisons des estimations $\hat{\tau}(v \to e')$ et $\hat{\tau}(s' \to v)$ plutôt que des durées exactes. La durée estimée est basée sur la distance directe entre deux emplacements v et e' (resp. s' et v) en utilisant la formule de Haversine. Il est à noter que pour un nœud donné v avec $\tau_d(s' \to v) + \tau_d(v \to e') > \tau_d(s' \to e') + \Delta_d$, cela ne nous permet pas de rejeter la potentialité du nœud v. Cela est dû au fait qu'il peut y avoir un autre nœud v', où v' est un lieu potentiel de dépose et que $t_o(v \to v') + t_d(v' \to e') < \tau_d(v \to e')$.

La formule Haversine (aussi connu sous le nom de "*distance orthodromique*") estime la distance entre deux emplacements en fonction de leur latitude/longitude. Cette formule utilise un modèle sphérique pour estimer la distance entre deux points sur la surface de la terre. Plus précisément, pour deux emplacements donnés $x = (\theta_1, \lambda_1)$ et $y = (\theta_2, \lambda_2)$ où θ_i, $i = 1, 2$, est la latitude, et λ_i, $i = 1, 2$, est la longitude. La distance entre x et y est donnée par $\hat{d}(x, y) = 2 \times R \times arcsin(\sqrt{a})$ où $R \approx 6371(km)$ est le rayon de la terre en kilomètre, et $a = sin^2(\frac{\theta_2 - \theta_1}{2}) + cos(\theta_1) \times cos(\theta_2) \times sin^2(\frac{\lambda_2 - \lambda_1}{2})$. Ainsi, la plus petite durée estimée de x à y est notée par $\hat{\tau}(x \to y) = \frac{\hat{d}(x,y)}{v_{max}}$, de telle sorte que v_{max} est la vitesse maximale tolérée parmi toutes les modalités utilisées par le conducteur et le passager dans la zone concernée. Il est possible d'utiliser d'autres estimations comme la projection cartésienne. Cette dernière est légèrement plus rapide à calculer mais moins précise en terme d'estimation.

Les ensembles définis ci-dessus correspondent à l'approche *a priori*, d'où la présence de ε dans les contraintes définies dans les deux ensembles $N^{\uparrow}(s)$ et $N^{\downarrow}(e)$. Pour les ensembles associés à l'approche *a posteriori* :

- Dans les deux ensembles $N^{\uparrow}(s)$ et $N^{\downarrow}(e)$, le coût du trajet commun est pris en charge en totalité par le passager (c'est à dire, $c_o(v \to e) = c_o(s \to v) = 0$).
- Inversement dans les deux ensembles du passager $N^{\uparrow}(s')$ et $N^{\downarrow}(e')$, le coût du trajet commun est pris en charge par le conducteur (c'est à dire, $\hat{d}(v \to e') = \hat{d}(s' \to v) = 0$).

Autrement dit, les ensembles seront définis comme suit :

$$N^{\uparrow}(s) = \{v \in V \mid c_o(s \to v) \leq c_o(s \to e) \ \wedge \ \tau_o(s \to v) + \tau_o(v \to e) \leq \tau_o(s \to e) + \Delta_o\}$$

$$N^{\downarrow}(e) = \{v \in V \mid c_o(v \to e) \leq c_o(s \to e) \ \wedge \ \tau_o(s \to v) + \tau_o(v \to e) \leq \tau_o(s \to e) + \Delta_o\}$$

$$N^{\uparrow}(s') = \{v \in \mathrm{V} \mid c_d(s' \to v) \leq c_d(s' \to e') \ \wedge \ \tau_d(s' \to v) + \hat{\tau}(v \to e') \leq \tau_d(s' \to e') + \Delta_d\}$$

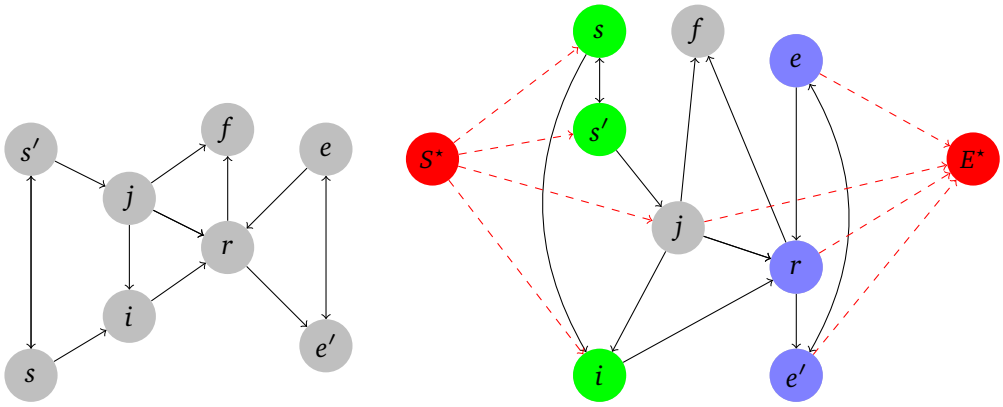
$$N^{\downarrow}(e') = \{v \in V \mid c_d(v \to e') \leq c_d(s' \to e') \ \wedge \ \hat{\tau}(s' \to v) + \tau_d(v \to e') \leq \tau_d(s' \to e') + \Delta_d\}$$

Le lemme suivant caractérise simultanément l'ensemble des lieux intermédiaires potentiels pour le conducteur et le passager.

Lemme. 2 *Un noeud $v \in V$ est un noeud intermédiaire potentiel si et seulement si $v \in C = C_1 \cup C_2$ où $C_1 = N^{\uparrow}(s) \cap N^{\uparrow}(s')$ et $C_2 = N^{\downarrow}(e) \cap N^{\downarrow}(e')$.*

3.3.2 Graphe transformé *G'*

Après avoir déterminé les deux classes potentielles de prise en charge C_1 et de dépose C_2, nous procédons à la transformation du graphe initial G en un nouveau graphe $G' = (V', A')$ constitué de deux classes et de deux nœuds artificiels. Plus précisément, nous ajoutons au graphe G deux nœuds artificiels : $S^{\star}$ et $E^{\star}$. Le nœud $S^{\star}$ est relié à chaque nœud v, $v \in C_1$ avec un arc $(S^{\star}, v)$ et un coût de $c(S^{\star}, v) = c_o(s \to v) + c_d(s' \to v)$. Le nœud $E^{\star}$ est relié à chaque nœud v, $v \in C_2$ avec un arc $(v, E^{\star})$ et un coût de $c(v, E^{\star}) = c_o(v \to e) + c_d(v \to e')$. L'arc $(S^{\star}, v)$ représente le déplacement du conducteur et du passager de leur position de départ jusqu'au lieu de prise en charge v. En revanche, l'arc $(v, E^{\star})$ représente le déplacement du passager et du conducteur du lieu de dépose v à leur destination respective. Dans la figure 3.3 nous présentons un exemple du graphe transformé.



(a) Graphe initial G (b) Graphe transformé G'

FIGURE 3.3 – La sous-figure (a) représente un exemple d'un graphe initial G constitué de 8 nœuds. La sous-figure (b) représente le nouveau graphe transformé G' qui est constitué de deux classes : C_1 (nœuds de couleur vert) et C_2 (nœuds de couleur bleu). Le nœud f n'appartient ni à la classe C_1 ni à la classe C_2, tandis que le nœud j appartient simultanément à ces deux deux classes.

Toutes les approches de résolution décrites dans ce chapitre sont polynomiales et elles sont basées sur le nouveau graphe G'.

3.3.3 Approche énumérative

Avec deux lieux intermédiaires : de prise en charge et le dépose, il est facile de vérifier si ces derniers forment *un appariement raisonnable pour le système a priori ou le système a posteriori*. L'approche énumérative liste toutes les paires possibles de lieux intermédiaires (r_1, r_2) dans C en sélectionnant la paire (r_1^*, r_2^*) avec un *trajet-total* minimal sous les contraintes d'appariement décrites dans la **définition 5** (pour le système a priori) ou la **définition 6** (pour le système a posteriori).

Algorithme 1 Méthode énumérative

Entrée: Graphe $G(V,A)$, lieux de départ s et s', les destinations e et e'.
Sortie: Paire de lieux intermédiaires (r_1^*,r_2^*), *trajet-total* $GlobalPath_{min}(s,s',r_1^*,r_2^*,e,e')$.

Initialisation : $GlobalPath_{\min} \leftarrow \infty$.
Déterminer la classe C_1 en utilisant deux algorithmes de Dijkstra-inverse un-vers-plusieurs (forward one-to-all Dijkstra), avec s et s' comme noeuds sources.
Déterminer la classe C_2 en utilisant deux algorithmes de Dijkstra-inverse un-vers-plusieurs (backward one-to-all Dijkstra), avec e et e' comme noeuds sources.
Pour tout r_1 **dans** C_1 **Faire**
    Calculer les coûts $c_d(r_1 \rightarrow r_2), \forall r_2 \in C_2$, en utilisant l'algorithme de Dijkstra un-vers-plusieurs avec r_1 comme source.
    Pour tout r_2 **dans** C_2 **Faire**
        Calculer $GlobalPath(s,s',r_1,r_2,e,e')$ comme décrit dans l'équation (3.1)
        Si $GlobalPath(s,s',r_1,r_2,e,e') < GlobalPath_{\min}$ **et** $r_1 \neq r_2$ **et** les contraintes d'appariement sont satisfaites **Alors**
            $GlobalPath_{\min} \leftarrow GlobalPath(s,s',r_1,r_2,e,e')$.
            $(r_1^*, r_2^*) \leftarrow (r_1, r_2)$.
        Fin Si
    Fin Pour
Fin Pour

La complexité temporelle de l'approche énumérative est de $|C_1|$ fois l'agorithme de Dijkstra de type un-vers-plusieurs, avec $|C_1|$ le cardinal de la classe.

3.3.4 Approches heuristiques pour le système de covoiturage a priori

Nous proposons deux heuristiques efficaces pour la sélection de lieux intermédiaires de prise en charge et de dépose dont le taux de partage de coûts sur le trajet commun est fixé a priori par le conducteur. Ces lieux seront déterminés de manière à ce que l'objectif (3.1) soit minimisé tout en respectant les contraintes d'appariement (3.3), (3.4), (3.5) et (3.6).
Pour ce faire, nous définissons deux métriques basées sur le coût du trajet et qui seront éventuellement utilisées pour séparer la classe C en deux sous-classes disjointes. Ainsi, la classe C_1 ne contiendra que des lieux potentiels de prise en charge et la classe C_2 ne contiendra que les lieux de dépose potentiels. Ensuite, nous calculons pour chaque nœud v dans C_2, le *trajet-total* optimal ayant le noeud v comme lieu de dépose. Une fois que les *trajets-totaux* ont été énumérés, nous sélectionnons le *trajet-total* avec le coût minimal qui satisfait les contraintes (3.5) et (3.6). Le partitionnement du graphe en deux classes disjointes assurera un chemin de $S^\star$ à $E^\star$ de cardinalité supérieure ou égale à trois.

Dans ce qui suit, nous présentons une description détaillée des étapes de nos deux heuristiques.

Métrique de partition simple ($\mathscr{M}_1$) La première métrique $\mathscr{M}_1$ partitionne les nœuds de $C_1 \cap C_2$ en les affectant à une des deux classes C_1 ou C_2 en fonction de leur positionnement par rapport aux lieux de départ s, s', et aux destinations e, e'. Ainsi, si un nœud est plus proche des deux lieux de départ que des deux destinations, alors ce nœud est retiré de la classe C_2. Autrement, ce nœud est retiré de la classe C_1. Plus formellement, pour un noeud $v \in C_1 \cap C_2$, si le coût $c_o(s \to v) + c_d(s' \to v) \leq c_o(v \to e) + c_d(v \to e')$ alors v est retiré de la classe C_2, sinon v est retiré de la classe C_1 (voir la figure 3.4).

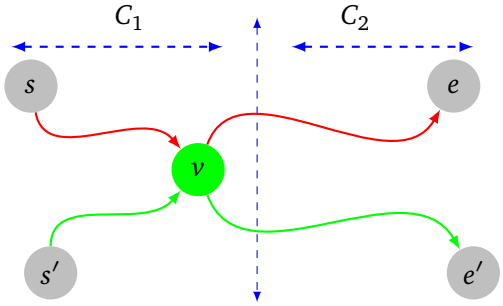


FIGURE 3.4 – Partitionnement d'un noeud v selon la métrique $\mathscr{M}_1$. Le noeud v est proche des emplacements de départ du passager et du conducteur, donc il est classé dans C_1 et retiré de C_2. Les lignes rouges correspondent à la trajectoire du conducteur, tandis que les lignes vertes correspondent à la trajectoire du passager.

Métrique de partition améliorée ($\mathscr{M}_2$) La deuxième métrique $\mathscr{M}_2$ améliore la qualité de la métrique $\mathscr{M}_1$ en estimant le coût du *trajet-total* de s, s' à e, e'. Plus formellement, pour un nœud $v \in C_1 \cap C_2$, si $c_o(s \to v) + c_d(s' \to v) + \alpha \leq c_o(v \to e) + c_d(v \to e') + \beta$, alors v est retiré de la classe C_2, sinon v est retiré de la classe C_1, où α et β sont définis comme suit :

$$\alpha = \max \begin{cases} \min\limits_{k \neq v} c_o(v,k) + \min\limits_{r \neq v} \left\{ c_o(r \to e) + c_d(r \to e') \right\} & \text{(5a)} \\ \min \begin{cases} c_o(v \to e') + c_o(e' \to e) & \text{(5b)} \\ c_o(v \to e) + \min\limits_{r \neq v} c_d(r, e') & \text{(5c)} \end{cases} \end{cases}$$

$$\beta = \max \begin{cases} \min\limits_{r \neq v} \left\{ c_o(s \to r) + c_d(s' \to r) \right\} + \min\limits_{k \neq v} c_o(k, v) & \text{(6a)} \\ \min \begin{cases} c_o(s \to s') + c_o(s' \to v) & \text{(6b)} \\ c_o(s \to v) + \min\limits_{r \neq v} c_d(s', r) & \text{(6c)} \end{cases} \end{cases}$$

Si v est un lieu de prise en charge, alors α est une borne inférieure parmi tous les chemins du conducteur et du passager à partir du lieu de prise en charge v jusqu'à leur destination avec au moins un trajet commun. Il est facile de voir que si $c_o(v \to u) \geq \min_{k \neq v} c_o(v,k)$ et
$c_o(u \to e) + c_d(u \to e') \geq \min_{r \neq v} \left\{ c_o(r \to e) + c_d(r \to e') \right\}$, alors l'équation (5a) est une borne inférieure valide.
Dans les équations (5b) et (5c), le lieu de dépose est respectivement la destination e' et un nœud r qui appartient au plus court chemin $v \to e$ (voir la figure 3.5). Tandis que β est une borne inférieure parmi tous les trajets du conducteur et du passager de leur emplacement de départ jusqu'au lieu de dépose v avec au moins un sous-chemin commun.

Les équations (6a), (6b) et (6c) sont respectivement des bornes inférieures symétriques aux équations (5a), (5b) et (5c) lorsque v est le lieu de dépose (voir la figure 3.6).

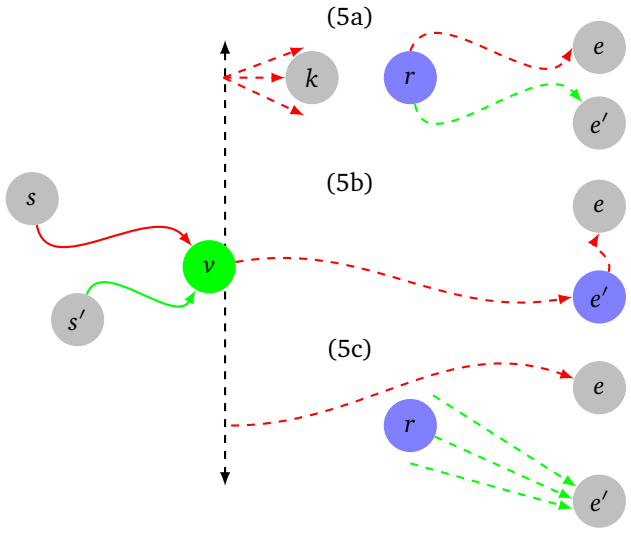


FIGURE 3.5 – Estimation du coût du *trajet-total* en considérant v comme un lieu de prise en charge. Les lignes de couleur rouge correspondent aux trajets du conducteur, tandis que les lignes de couleur verte correspondent aux trajets du passager. Les lignes en pointillé représentent des estimations du coût des trajets restants depuis le lieu de prise en charge v jusqu'à leur destination (c'est à dire la quantité α).

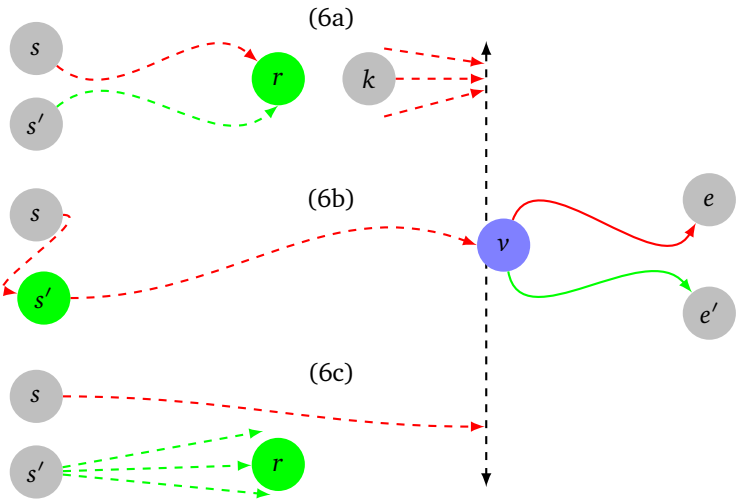
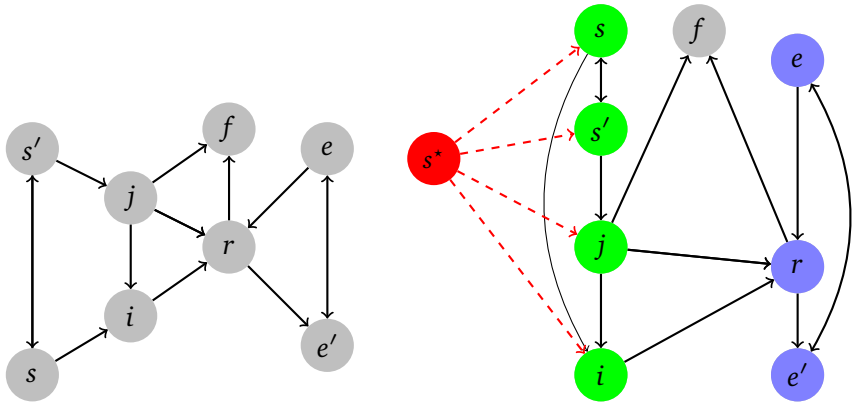


FIGURE 3.6 – Estimation du coût du *trajet-total* en considérant v comme un lieu de dépose. Les lignes de couleur rouge correspondent aux trajets du conducteur, tandis que les lignes de couleur verte correspondent aux trajets du passager. Les lignes en pointillé représentent des estimations du coût des trajets restants depuis leur lieu de départ jusqu'à leur lieu de dépose v (c'est à dire la quantité β).

Après le partitionnement disjoint des deux classes C_1 et C_2, l'étape suivante détermine le plus court *trajet-total* avec un lieu de dépose donné. Nous notons que le coût des arcs considérés entre les deux classes correspond au coût du conducteur. En effet, ces arcs correspondent aux trajets partagés

entre le passager et le conducteur (voir figure 3.7).



(a) Graphe initial G (b) Graphe transformé G'

FIGURE 3.7 – La sous-figure (a) représente un exemple d'un graphe initial G constitué de 8 nœuds. La sous-figure (b) représente le nouveau graphe transformé G' qui se compose de deux classes disjointes C_1 (noeuds de couleur vert) et C_2 (noeuds de couleur bleu). Le nœud f n'appartient ni à C_1 ni à C_2.

On note par $\Phi(v) = (S^\star, \phi(1), \ldots, \phi(k), v)$ le chemin dans le graphe G' où $S^\star$ et v sont respectivement le noeud initial et le noeud final. Ainsi, les éléments suivants sont vérifiés :

Fait 1 *Chaque chemin $\Phi(v) = (S^\star, \phi(1), \ldots, \phi(k), v)$ dans G' implique que $\phi(1) \in C_1$.*

Fait 2 *Étant donné un chemin optimal $\Phi(v) = (S^\star, \phi(1), \ldots, \phi(k), v)$, avec $v \in C_2$ dont le coût est $c(\Phi)$. Un trajet-total $GlobalPath(s, s', \phi(1), v, e, e')$ peut être dérivé de ce chemin où $\phi(1)$ et v correspondent respectivement à l'emplacement intermédiaire de prise en charge et de dépose avec $GlobalPath(s, s', \phi(1), v, e, e') = c(\Phi(v)) + c_o(v \to e) + c_d(v \to e')$.*

Le corollaire suivant en est une conséquence directe.

Corollaire. 1 *Soit Ω l'ensemble des plus courts trajets-totaux tel que pour chaque v dans C_2, un plus court trajet-total ayant v comme lieu de dépose est ajouté à Ω. L'ensemble Ω peut être déterminé avec une complexité temporelle de $O(|V'| \log |V'| + |A'|)$ en utilisant l'algorithme de Dijkstra un-vers-plusieurs avec un tas de Fibonacci depuis $S^\star$ vers tous les nœuds C_2 dans le graphe G'.*

Enfin, la dernière étape sélectionne la solution avec un coût minimal du *trajet-total* qui satisfait les contraintes (3.5) et (3.6). L'approche détaillée est donnée dans l'algorithme 2.

Algorithme 2 Approche heuristique selon la métrique $\mathscr{M}_i$

Entrée: Graphe $G(V,A)$, lieux de départ s et s', destinations e et e'.
Sortie: Lieux intermédiaires r_1^* et r_2^*, *trajet-total* $GlobalPath(s,s',r_1^*,r_2^*,e,e')$.

Initialisation : $C \leftarrow \emptyset$, $C_1 \leftarrow \emptyset$ et $C_2 \leftarrow \emptyset$.
Calculer deux algorithmes de Dijkstra-inverse un-vers-plusieurs avec e et e' comme noeuds cibles.
Calculer deux algorithmes de Dijkstra un-vers-plusieurs avec s et s' comme noeuds sources.
Calculer un algorithme de Dijkstra un-vers-plusieurs depuis s' avec le coût conducteur.
Calculer un algorithme de Dijkstra inverse un-vers-plusieurs depuis e' avec le coût conducteur.
Pour tout r **dans** V **Faire**
  Si $r \in N^{\uparrow}(s)$ **et** $r \in N^{\uparrow}(s')$ **Alors**
    $C_1 \leftarrow C_1 \cup \{r\}$.
  Fin Si
  Si $r \in N^{\downarrow}(e)$ **et** $r \in N^{\downarrow}(e')$ **Alors**
    $C_2 \leftarrow C_2 \cup \{r\}$.
  Fin Si
  Si $r \in C_1 \cap C_2$ **Alors**
    Tester la condition de la métrique $\mathscr{M}_i$ et retirer r de la classe appropriée C_1 ou C_2
  Fin Si
Fin Pour
Ajouter le nœud $S^\star$ au graphe G et relier $S^\star$ aux noeuds de la classe C_1 avec un coût $c(S^\star,u) \leftarrow c_o(s \rightarrow u) + c_d(s' \rightarrow u)$.
Calculer un algorithme de Dijkstra un-vers-plusieurs sur le nouveau graphe G' depuis $S^\star$ pour tous les noeuds dans C_2.
Sélectionner le plus court chemin dans le nouveau graphe G' satisfaisant les contraintes (3.5) et (3.6) avec deux lieux intermédiaires $r_1^* \in C_1$ et $r_2^* \in C_2$.

Complexité. Afin de réduire le temps de calcul de nos heuristiques, nous commençons tout d'abord par l'exécution de 2 algorithmes de Dijkstra-inverse de type un-vers-plusieurs avec e et e' comme noeuds cibles, auquel s'ajoute deux autres algorithmes de Dijkstra un-vers-plusieurs avec s et s' comme noeuds sources. Ensuite, nous calculons deux algorithmes de Dijkstra un-vers-plusieurs avec le coût conducteur. Le premier depuis s' et le second sur le graphe inversé G^{-1} depuis e'. Ces calculs permettent de déterminer la classe C en calculant les ensembles $N^{\uparrow}(s)$, $N^{\uparrow}(s')$, $N^{\downarrow}(e)$, $N^{\downarrow}(e')$ ainsi que le coût des arcs de A'. Ces étapes sont déterminées en $O(6(|V|\log|V|+|A|))$. Cependant, le partitionnement effectué par les deux métriques $\mathscr{M}_1$ et $\mathscr{M}_2$ est donné en $O(|V|)$. Ainsi, l'ensemble des *trajet-totaux* sera déterminé en calculant un algorithme de Dijkstra un-vers-plusieurs de $S^\star$ vers tous les noeuds de C_2 en $O(|V'|\log|V'|+|A'|)$. Enfin, nous sélectionnons le meilleur *trajet-total* qui satisfait les contraintes (3.5) et (3.6). Cette sélection peut se faire parallèlement lors de la détermination des *trajet-totaux*. La complexité de cette heuristique est de $O(6(|V|\log|V|+|A|)+|V|+(|V'|\log|V'|+|A'|))$. Dans le cas où le conducteur et le passager ont le même coût, la complexité se réduit à $O(4(|V|\log|V|+|A|)+|V|+(|V'|\log|V'|+|A'|))$. Dans ce qui suit, les heuristiques utilisant la métrique $\mathscr{M}_1$ et $\mathscr{M}_2$ sont respectivement notées par HM_1 et HM_2.

3.3.5 Approches heuristiques pour le système de covoiturage a posteriori

Dans ce qui suit, nous décrivons deux approches pour le système de covoiturage a posteriori. La première approche repose sur une recherche bidirectionnelle et la seconde approche est basée sur le plus court chemin un-vers-plusieurs. Dans ce système, le partitionnement de la classe potentielle C en deux classes disjointes C_1 et C_2 n'est plus nécessaire. Autrement dit, un nœud appartenant simultanément aux deux classes C_1 et C_2 ne sera plus départagé. Ce dernier est automatiquement

géré par notre procédure de *relaxation d'arcs* qui garantit le fait que la contrainte de réduction de coût (3.7) n'est pas réalisable si le chemin résultant du nouveau graphe G' est seulement composé de trois nœuds. Il à noter que le coût de partage dans le système a posteriori n'est pas fixé a priori par les utilisateurs et que le gain généré en covoiturage est directement assuré par la contrainte (3.7).

Procédure de relaxation d'arcs modifiée

Dans un algorithme de type Dijkstra. Lorsqu'un nœud v est réglé, pour chaque arc sortant (v,u), nous testons si via cet arc, le nœud u peut être atteignable avec un coût inférieur ou égal au coût actuel de u. Si cela est possible, le coût provisoire du nœud u est mis à jour à la nouvelle valeur. Cette procédure est appelée la procédure de *relaxation d'arcs*. Notre procédure de *relaxation d'arcs* diffère de celle de la littérature [Dijkstra, 1959]. Cette différence réside simplement dans la mise à jour forcée du coût du nœud v même si le nouveau coût est égal au coût actuel du nœud. En effet, cette modification permet de nous assurer que si le plus court chemin de $S^\star$ à $E^\star$ ne contient qu'un seul nœud intermédiaire, c'est à dire $S^\star \overset{arc}{\rightsquigarrow} r_1 \overset{arc}{\rightsquigarrow} E^\star$, alors le conducteur et le passager ne seront pas en mesure de former un appariement raisonnable. Plus précisément, la contrainte (3.7) ne sera pas respectée. Ce cas peut se produire uniquement lorsque le plus court chemin du conducteur et celui du passager se croisent en un seul point. Enfin, dans le chemin $S^\star \rightarrow E^\star$ (i.e. $S^\star \overset{arc}{\rightsquigarrow} r_1^\star \rightarrow r_2^\star \overset{arc}{\rightsquigarrow} E^\star$), nous retrouvons le meilleur lieu de prise en charge $r_1^\star$ et le meilleur lieu de dépose $r_2^\star$ qui correspondent respectivement au successeur de $S^\star$ et au prédécesseur de $E^\star$.
La figure 3.8 présente un exemple de plus court chemin de $S^\star$ à $E^\star$ en utilisant notre procédure de *relaxation d'arcs*.

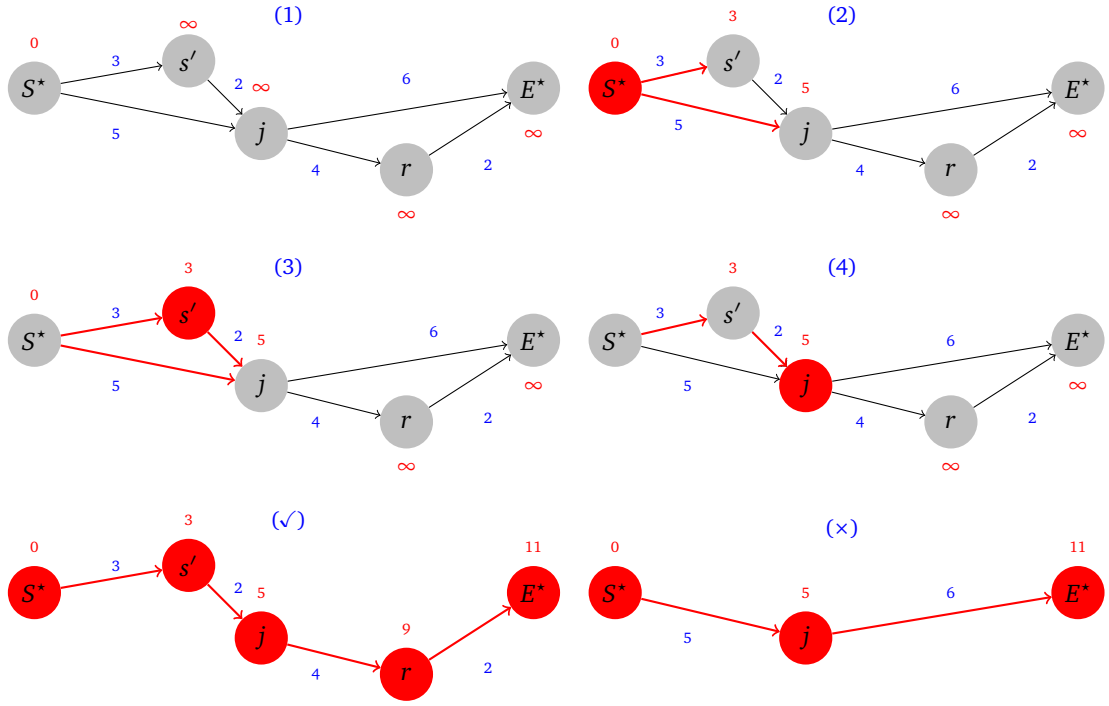


FIGURE 3.8 – Plus court chemin de $S^\star$ à $E^\star$ en utilisant notre approche de *relaxation d'arcs modifiée*.

Initialement, tous les nœuds à l'exception du nœud source sont étiquetés avec un coût provisoire égal à l'infini. Ensuite, les coûts provisoires des deux voisins s' et j sont mis à jour respectivement à 3

et à 5. Deuxièmement, une fois que le nœud $S^\star$ est réglé, nous sélectionnons le nœud avec le plus petit coût provisoire, c'est à dire s'. A ce niveau, le chemin pour atteindre le nœud j (voisin de s') doit être mis à jour à $S^\star \overset{arc}{\rightsquigarrow} s' \overset{arc}{\rightsquigarrow} j$. Cette mise à jour est effectuée malgré l'égalité des coûts des deux chemins, le premier vient directement de $S^\star$ et le deuxième passe par s'. Le même raisonnement s'applique à chaque nœud. Enfin, nous obtenons un chemin composé de 5 nœuds ($S^\star \overset{arc}{\rightsquigarrow} s' \overset{arc}{\rightsquigarrow} j \overset{arc}{\rightsquigarrow} r \overset{arc}{\rightsquigarrow} E^\star$) plutôt que le chemin composé de 3 nœuds ($S^\star \overset{arc}{\rightsquigarrow} j \overset{arc}{\rightsquigarrow} E^\star$). Le chemin composé de 3 nœuds comporte un seul nœud intermédiaire.
Les deux approches proposées ci-dessous utilisent la procédure modifiée de *relaxation d'arcs*.

Recherche Bidirectionnelle Modifiée - RBM

L'algorithme RBM fonctionne avec deux recherches simultanées du plus court chemin. La première recherche depuis le nœud source $S^\star$ et la seconde recherche depuis la cible $E^\star$ en même temps jusqu'à ce que "les deux frontières de recherche se rencontrent". Plus précisément, l'algorithme RBM maintient deux files de priorité, l'une pour la recherche avant (forward search) à partir de la source $S^\star$ notée par Q_1, l'autre pour une recherche arrière (backward search) à partir de la cible $E^\star$ notée Q_2. Cette dernière est une recherche simple vers l'avant sur le graphe inversé G^{-1} dans lequel chaque arc (v,u) est remplacé par (u,v) dans G^{-1}.
Dans chaque itération, nous récupérons le nœud avec le plus petit coût provisoire, soit depuis la source $S^\star$ ou bien depuis la cible $E^\star$. Pour ce faire, une simple comparaison des deux files de priorité sur le minimum Q_1 et Q_2 suffit. Une fois qu'un nœud v est réglé dans une file de priorité qui est déjà réglée dans l'autre file , nous obtenons le premier coût provisoire du plus court chemin de $S^\star$ à $E^\star$. Le coût correspondant à ce dernier est le coût du chemin trouvé par la *recherche avant* de $S^\star$ à v plus le coût du chemin trouvé par la *recherche arrière* de v à $E^\star$.
L'intersection des deux recherches dans un nœud ne garantit pas l'optimalité du plus court chemin trouvé. Pour garantir l'optimalité, les deux recherches doivent se poursuivre jusqu'à ce que le coût provisoire minimal des deux files de priorité soit supérieur au coût du plus court chemin provisoire (ce qui correspondra enfin au plus court chemin). A chaque itération, lorsque le coût provisoire du plus court chemin $S^\star$ à $E^\star$ est mis à jour, nous vérifions la validité des contraintes (3.5), (3.6) et (3.7). Finalement, nous gardons le chemin admissible qui minimise *le trajet-total*.

Plus Court Chemin un-vers-plusieurs Modifié - PCCM

Dans cette méthode, au lieu de calculer un plus court chemin de $S^\star$ à $E^\star$, nous calculons pour chaque nœud v dans C_2 le plus court chemin $S^\star \rightarrow v$. Cette approche nous permet de trouver tous les chemins qui satisfont la contrainte (3.7). Une fois que tous les plus courts chemins satisfaisant la contrainte de (3.7) ont été énumérés, nous sélectionnons le *trajet-total* avec un coût minimal qui satisfait les contraintes (3.5) et (3.6).

Le principal avantage de la première méthode (**RBM**) réside dans le temps de calcul qui est moins important que la seconde méthode (**PCCM**). En effet, cette dernière exige le calcul de tous les plus courts chemins vers chaque nœud dans la classe C_2. D'autre part, la deuxième méthode a un plus grand contrôle sur les contraintes du *temps de détour* pour le conducteur et le passager.

Complexité. Les deux classes C_1 et C_2 sont déterminées de la même façon que le système *a priori*. Une fois que ces deux dernières sont définies, on procède directement au calcul du plus court chemin sans aucune partition disjointe des deux classes. Plus spécifiquement, pour la première approche **RBM** on utilise l'algorithme de Dijkstra bidirectionnel modifié depuis la source $S^\star$ vers la destination

$E^\star$ en gardant le chemin qui respecte les contraintes d'appariement. En revanche, pour la seconde approche **PCCM**, on utilise l'algorithme de Dijkstra un-vers-plusieurs depuis la source $S^\star$ vers les nœuds de la classe C_2. Une fois que les chemins vers tous les noeuds de la classe C_2 sont découverts, on sélectionne le meilleur chemin satisfaisant les contraintes d'appariement.

3.4 Problème de sélection du meilleur conducteur

Les approches développées dans la section précédente correspondent au cas où une offre et une demande sont déjà fixées à l'avance. Dans un cadre dynamique, les demandes et les offres de covoiturage arrivent continuellement à des moments arbitraires. Par conséquent, les appariements entre les passagers et les conducteurs doivent être calculés de nombreuses fois au cours de la journée car les futures demandes sont inconnues. Dans cette section, nous considérons le problème de sélection du meilleur conducteur. Spécifiquement, sur un ensemble d'offres de conducteurs déjà dans le système, lorsqu'une requête d'un passager d entre dans le système, l'objectif est de sélectionner le conducteur qui sera en mesure de satisfaire au mieux la demande du passager en se basant sur un seul emplacement intermédiaire.

Plusieurs critères de sélection peuvent être envisagés afin de trouver le meilleur conducteur. Par exemple, nous pouvons choisir le conducteur qui permet au passager d'arriver le plus tôt possible à sa destination et/ou qui minimise son temps de trajet total. Le premier objectif est en conflit avec le deuxième. En effet, le conducteur qui emmène le passager le plus rapidement possible à sa destination n'est pas nécessairement le conducteur qui minimise le temps du trajet total.

La règle de sélection nécessite le stockage de certaines informations pour chaque offre entrant dans le système : lorsqu'une offre o_i est ajoutée au système, nous stockons tous les emplacements intermédiaires admissibles pour cette offre. Ainsi, nous déterminons les coûts $c_o(s_i \rightarrow v)$ et $c_o(v \rightarrow e_i)$. La procédure d'ajout d'une offre est détaillée ci-dessous.

3.4.1 Ajout d'une offre

Chaque nouvelle offre o_i arrivant dans le système génère de nouvelles possibilités de covoiturage. Lorsque cette dernière est ajoutée dans le système, nous stockons tous les emplacements intermédiaires possibles pour lesquels le conducteur peut passer sans enfreindre la contrainte du temps de détour en déterminant également les coûts $c_o(s_i \rightarrow v)$ et $c_o(v \rightarrow e_i)$. Pour ce faire, nous notons par $M^{\uparrow}(s_i)$ et $M^{\downarrow}(e_i)$ l'espace de *recherche avant* (forward search space) et *l'espace de recherche arrière* (backward search space). L'espace de *recherche avant* $M^{\uparrow}(s_i)$ est un ensemble de triplets : nœud, coût et durée $(v, c^{\uparrow}_{s_i}, \tau^{\uparrow}_{s_i})$ tel qu'il existe un chemin de s_i à v avec un coût $c^{\uparrow}_{s_i}$ et une durée $\tau^{\uparrow}_{s_i}$. De l'autre côté, *l'espace de recherche arrière* $M^{\downarrow}(e_i)$ est un ensemble de triplets : nœud, coût et durée $(v, c^{\downarrow}_{e_i}, \tau^{\downarrow}_{e_i})$ tel qu'il existe un chemin de v à e_i avec un coût $c^{\downarrow}_{e_i}$ et une durée $\tau^{\downarrow}_{e_i}$.
En se basant sur les contraintes du temps de détour, l'espace de recherche peut être limité sans considérer aucune demande au préalable. Dans notre approche, nous commençons par déterminer l'ensemble $M^{\downarrow}(e_i)$ en utilisant l'algorithme de Dijkstra inverse [Dijkstra, 1959], et pour chaque nœud v marqué, nous vérifions la validité de la contrainte $\tau_o(v \rightarrow e_i) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_i)$. La procédure de recherche est stoppée pour le premier nœud marqué v qui ne respecte pas la précédente contrainte. Dans la seconde étape, nous déterminons l'espace de *recherche avant* $M^{\uparrow}(s_i)$, en utilisant l'algorithme de Dijkstra. Dans ce cas, la procédure de recherche est stoppée pour le premier nœud marqué qui ne respecte pas la contrainte $\tau_o(s_i \rightarrow v) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_i)$. Finalement, nous gardons un nœud v dans $M^{\uparrow}(s_i)$ seulement si $\big(\tau_o(s_i \rightarrow v) + \tau_o(v \rightarrow e_i)\big) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_i)$.
A chaque itération lorsqu'un nœud v correspondant à un emplacement intermédiaire est sélectionné

dans l'ensemble $M^{\uparrow}(s_i)$, nous ajoutons quelques informations dans le *seau* $Bucket_C(v)$. Ce dernier est définit comme suit ;

Definition. 8 ***(Bucket des conducteurs*** $Bucket_C(v)$, $v \in V$*)*
Le bucket des conducteurs $Bucket_C(v)$ *associé à un emplacement* v *est l'ensemble de quintuplets composés de l'identifiant d'un conducteur* i*, la distance nécessaire pour atteindre l'arrêt* v *à partir de sa position actuelle, l'heure d'arrivée à l'emplacement* v*, la distance et la durée restante pour atteindre sa destination finale depuis l'emplacement* v*, c'est-à-dire ;*

$$Bucket_C(v) := \{(i, c^{\uparrow}_{s_i}, \tau^{\uparrow}_{s_i}, c^{\downarrow}_{e_i}, \tau^{\downarrow}_{e_i}) \quad | \quad (\tau^{\uparrow}_{s_i} + \tau^{\downarrow}_{e_i}) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_i)\}. \tag{3.10}$$

Pour chaque emplacement intermédiaire v on associe un *bucket*. Ce dernier sert à stocker tous les trajets qui passent via cet emplacement sans enfreindre la borne supérieure de la contrainte du temps de détour.
Pour prendre en considération les conducteurs ayant déjà entamé leur voyage, on doit actualiser leur position à chaque unité de temps. Pour simplifier, nous considérons seulement les conducteurs qui n'ont pas encore entamé leur voyage lorsqu'une demande de covoiturage arrive dans le système. Les deux algorithmes de Dijkstra utilisés dans les deux espaces de recherche $M^{\uparrow}(s_i)$ et $M^{\downarrow}(e_i)$ pourront être guidés par une fonction qui permet d'estimer les durées restantes soit de v vers e_i pour l'ensemble $M^{\uparrow}(s_i)$, ou bien depuis s_i vers v pour l'ensemble $M^{\downarrow}(e_i)$.

3.4.2 Suppression d'une offre

La suppression des offres est nécessaire lorsqu'un trajet a été achevé ou bien dans le cas où un conducteur souhaite retirer l'offre. Pour supprimer une offre o_i, nous devons retirer ses entrées stockées dans les buckets. Afin d'accélérer le temps de calcul de la suppression, nous classons les buckets par jour de passage. Autrement dit, pour chaque jour du calendrier, l'ordre dans lequel les entrées sont insérées dans un bucket n'est pas défini. Cela rend l'ajout d'une offre très rapide, mais le retrait, nécessite la numérisation des buckets. Cependant, la numérisation de tous les buckets est prohibitif dû aux nombres élevés d'entrées. Au lieu de cela, il serait plus judicieux de calculer $M^{\uparrow}(s_i)$ et $M^{\downarrow}(e_i)$ afin d'obtenir l'ensemble des buckets qui contient une entrée de cette offre o_i. Puis, nous n'avons qu'à parcourir ces buckets et enfin supprimer les entrées de l'offre o_i.

3.4.3 Méthode exacte pour la sélection de la meilleure offre et les deux emplacements intermédiaires de prise en charge et de dépose

La méthode exacte consiste à parcourir chaque couple de bucket $(Bucket_C(r_1), Bucket_C(r_2))$ dans $(N^{\uparrow}(s'), N^{\downarrow}(e'))$ afin de déterminer le meilleur appariement. Soit $\sigma^{\star}$ le coût optimal du *trajet-total* dont la valeur initiale est $\sigma^{\star} = +\infty$. Après la détermination des deux ensembles $N^{\uparrow}(s')$ et $N^{\downarrow}(e')$, nous parcourons les noeuds de l'ensemble $N^{\uparrow}(s')$. Pour chaque lieu intermédiaire v_1 de l'ensemble $N^{\uparrow}(s')$, nous calculons le coût $c_o(v_1 \rightarrow v_2)$ en utilisant l'algorithme de Dijkstra un-vers-plusieurs (forward one-to-all Dijkstra algorithm) pour tout nœud $v_2 \in N^{\downarrow}(e')$. Pour chaque offre o_i telle que $(i, c^{\uparrow}_{s_i}, \tau^{\uparrow}_{s_i}, c^{\downarrow}_{e_i}, \tau^{\downarrow}_{e_i}) \in Bucket_C(v_1)$, nous parcourons chaque nœud v_2 de l'ensemble $N^{\downarrow}(e')$. Ainsi, lorsque ces deux conditions suivantes sont vérifiées : (1) i est dans l'ensemble $Bucket_C(v_2)$ tel que o_i et d forment un appariement raisonnable avec v_1 (v_2) comme lieu de prise en charge (de dépose), et (2) le coût du *trajet-total* est inférieur à $\sigma^{\star}$, alors nous mettrons à jour le coût optimal du *trajet-total* à $GlobalPath(s, s', v_1, v_2, e, e')$. Ce processus est répété jusqu'à ce que tous les nœuds de l'ensemble $N^{\uparrow}(s')$ soient examinés. Les différentes étapes sont données dans l'algorithme 3. Il est à noter que cette méthode est applicable pour les deux systèmes (a *priori* et a *posteriori*) en prenant en compte les contraintes correspondantes au système choisi.

Algorithme 3 Méthode exacte de sélection

Entrée: Demande d, ensembles $N^{\uparrow}(s')$, $N^{\downarrow}(e')$, et $Bucket_C(v), \forall v \in G$.
Sortie: Meilleur conducteur $i^{\star}$, *trajet-total* $\sigma^{\star}$.

```
1: Initialisation : i* ← −1, σ* ← +∞.
2: Pour tout v_1 dans N↑(s') Faire
3:    Calculer les coûts c_o(v_1 → v_2), ∀ v_2 ∈ N↓(e') en utilisant l'algorithme de Dijkstra un-vers-plusieurs
      (forward one-to-all Dijkstra algorithm).
4:    Pour tout i dans Bucket_C(v_1) Faire
5:       Si v_1 forme une synchronisation temporelle pour i et d Alors
6:          Pour tout v_2 dans N↓(e') Faire
7:             Si i ∈ Bucket_C(v_2) et le conducteur i et le passager d forment un appariement raisonnable avec
               v_1 comme lieu de prise en charge et v_2 comme lieu de dépose Alors
8:                Si σ* > GlobalPath(s_i, s', v_1, v_2, e_i, e') Alors
9:                   i* ← i
10:                  σ* ← GlobalPath(s_i, s', v_1, v_2, e_i, e')
11:               Fin Si
12:            Fin Si
13:         Fin Pour
14:      Fin Si
15:   Fin Pour
16: Fin Pour
```

Complexité temporelle. La complexité temporelle de l'algorithme de Dijkstra en utilisant le tas de Fibonacci est de $O(|V|\log|V|+|E|)$. Les deux ensembles $N^{\uparrow}(s')$, $N^{\downarrow}(e')$ peuvent être déterminés en $O(2\cdot(|V|\log|V|+|E|))$. Cependant, l'algorithme 3 s'exécute en $O(|N^{\uparrow}(s')|(|V|\log|V|+|E|)+|N^{\downarrow}(e')|\cdot \sum_{v_1\in N^{\uparrow}(s')}|Bucket_C(v_1)|)$.

3.4.4 Heuristique de sélection d'offre

Même si la méthode exacte de sélection fournit une solution en temps polynomial, le temps de calcul devient assez long pour les grandes instances sur un vrai réseau routier. Dans un contexte de temps réel, la solution doit être déterminée en quelques secondes. Dans cette section, nous proposons une méthode heuristique avec une complexité temporelle inférieure qui permet de sélectionner une meilleure offre pour satisfaire la demande d.

L'heuristique de sélection est présentée par l'algorithme 6 composé de deux algorithmes 4 et 5. L'algorithme 4 nous permet de déterminer une offre qui minimise le *trajet-total* en faisant varier le lieu de prise en charge possible dans l'ensemble $N^{\uparrow}(s')$, tandis que le lieu de dépose est fixé à la destination du passager et du conducteur.

Les étapes 6 et 11 de l'algorithme 4 testent l'admissibilité du chemin trouvé en termes du temps de détour du conducteur et du passager ainsi que les coûts économisés. D'autre part, l'algorithme 5 détermine la meilleure offre qui minimise le *trajet-total*, en considérant les nœuds de l'ensemble $N^{\downarrow}(e')$ comme lieux potentiels de dépose, tandis que le lieu de prise en charge est fixé à l'emplacement de départ du conducteur et du passager. Enfin, la meilleure offre i^{*} correspond à l'une des offres déterminée par l'algorithme 4 et 5. Cette dernière est choisie de tel sorte que le gain généré soit le plus élevé. Afin d'améliorer le coût du *trajet-total* pour l'appariement choisi (i^{*}, d), nous utilisons l'une des deux méthodes décrites dans la section 3.3 (qui déterminera les deux meilleurs emplacements intermédiaires). Les détails des étapes sont présentés dans l'algorithme 6. Cette méthode permet de réduire la complexité à $O(k.n)$, où k est nombre d'offres se trouvant dans les buckets visités.

Algorithme 4 Parcours de l'ensemble $N^{\uparrow}(s')$

Entrée: Demande d, ensembles $N^{\uparrow}(s')$, $N^{\downarrow}(e')$ et $Bucket_C(v), \forall v \in G$.
Sortie: Le meilleur conducteur $i^{\star}$, *trajet-total* $\sigma^{\star}$.

Initialisation : $i^{\star} \leftarrow -1$, $\sigma^{\star} \leftarrow \infty$.
Calculer les coûts $c_o(v_1 \rightarrow e')$, $\forall\ v_1 \in N^{\uparrow}(s')$ en utilisant l'algorithme inverse de Dijkstra plusieurs-vers-un (backward one-to-all Dijkstra Algorithm).
Pour tout v_1 **dans** $N^{\uparrow}(s')$ **Faire**
  Pour tout i **dans** $Bucket_C(v_1)$ **Faire**
    Si v_1 forme une synchronisation temporelle entre le conducteur i et le passager d **Alors**
      Si $i \in Bucket_C(e')$ **et** $\tau_d(s' \rightarrow v_1) + \tau_o(v_1 \rightarrow e') \leq (\tau_d(s' \rightarrow e') + \Delta_d)$ **et** $\tau_o(s_i \rightarrow v_1) + \tau_o(v_1 \rightarrow e') + \tau_o(e' \rightarrow t_i) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_{o_i})$ **et** $c_o(s_i \rightarrow e_i) + c_d(s' \rightarrow e') - GlobalPath(s_i, s', v_1, e', e_i, e') \geq 0$ **Alors**
        Si $\sigma^{\star} > GlobalPath(s_i, s', v_1, e', e_i, e')$ **Alors**
          $i^{\star} \leftarrow i$, $\sigma^{\star} \leftarrow GlobalPath(s_i, s', v_1, e', e_i, e')$
        Fin Si
      Fin Si
      Si $\tau_d(s' \rightarrow v_1) + \tau_o(v_1 \rightarrow e_i) + \tau_d(e_i \rightarrow e') \leq (\tau_d(s' \rightarrow e') + \Delta_d)$ **et** $\tau_o(s_i \rightarrow v_1) + \tau_o(v_1 \rightarrow e_i) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_{o_i})$ **et** $c_o(s_i \rightarrow e_i) + c_d(s' \rightarrow e') - GlobalPath(s_i, s', v_1, e_i, e_i, e') \geq 0$ **Alors**
        Si $\sigma^{\star} > GlobalPath(s_i, s', v_1, e_i, e_i, e')$ **Alors**
          $i^{\star} \leftarrow i$, $\sigma^{\star} \leftarrow GlobalPath(s_i, s', v_1, e_i, e_i, e')$
        Fin Si
      Fin Si
    Fin Si
  Fin Pour
Fin Pour

Complexité. Les deux algorithmes 4 et 5 sont respectivement d'une complexité temporelle de $O(|V|\log|V| + |A| + \sum_{v_1 \in N^{\uparrow}(s')} |Bucket_C(v_1)|)$ et $O(|V|\log|V| + |A| + \sum_{v_2 \in N^{\downarrow}(e')} |Bucket_C(v_2)|)$. Ainsi, en ayant les deux ensembles $N^{\uparrow}(s')$, $N^{\downarrow}(e')$ et la meilleur offre $i^{\star}$, l'étape 4 dans l'algorithme 6 est donnée en $O\big((|V|\log|V|+|A|)+|N^{\uparrow}(s')|+|N^{\downarrow}(e')|\big)$. Finalement, l'heuristique de sélection réduit la complexité à $O\big(5\cdot(|V|\log|V|+|A|)+|N^{\uparrow}(s')|+|N^{\downarrow}(e')|+\sum_{v_1 \in N^{\uparrow}(s')} |Bucket_C(v_1)|+\sum_{v_2 \in N^{\downarrow}(e')} |Bucket_C(v_2)|\big)$.

3.4.5 Taux d'économie minimum

Dans certaines situations, le conducteur ou/et le passager peuvent exiger un taux minimum d'économie sur leur trajet en covoiturage. Par exemple, un conducteur peut exiger au moins 10% du gain par rapport à son coût s'il voyageait seul. Dans ce cas, nous étendons le modèle pour tenir compte de cette condition. Plus précisément, une offre et une demande de covoiturage seront respectivement représentées par $o = (s, e, [t_o^{\min}, t_o^{\max}], \Delta_o, \sigma_o)$ et $d = (s', e', [t_d^{\min}, t_d^{\max}], \Delta_d, \sigma_d)$, où σ_o (σ_d) est le pourcentage minimum de réduction des coûts fixé par le conducteur (passager) par rapport à son plus court chemin. Dans un tel cas, la contrainte sur l'économie des coûts (3.7) entre les deux participants est définie comme suit :

$$\begin{aligned} c_o(s \rightarrow e) + c_d(s' \rightarrow e') - (c_o(s \rightarrow r_1) + c_d(s' \rightarrow r_1) + c_o(r_1 \rightarrow r_2) + \\ c_o(r_2 \rightarrow e) + c_d(r_2 \rightarrow e')) \geq \sigma_o \cdot c_o(s \rightarrow e) + \sigma_d \cdot c_d(s' \rightarrow e') \end{aligned} \tag{3.11}$$

Les valeurs de ε_1 et ε_2 dans le lemme 1 sont réajustées afin de considérer cette nouvelle contrainte.

Algorithme 5 Parcours de l'ensemble $N^{\downarrow}(e')$

Entrée: Demande d, ensemble $N^{\uparrow}(s')$, ensemble $N^{\downarrow}(e')$ et bucket $Bucket_C(v), \forall v \in G$.
Sortie: Le meilleur conducteur $i^{\star}$, *trajet-total* $\sigma^{\star}$.

Initialisation : $i^{\star} \leftarrow -1$, $\sigma^{\star} \leftarrow \infty$.
Calculer les coûts $c_o(s' \rightarrow v_2)$, $\forall\ v_2 \in N^{\downarrow}(e')$ en utilisant l'algorithme de Dijkstra un-vers-plusieurs.
Pour tout v_2 **dans** $N^{\downarrow}(e')$ **Faire**
  Pour tout i **dans** $Bucket_C(v_2)$ **Faire**
    Si s' forme une synchronisation temporelle entre le conducteur i et le passager d **Alors**
      Si $i \in Bucket_C(s')$ **et** $\tau_o(s' \rightarrow v_2) + \tau_d(v_2 \rightarrow e') \leq (\tau_d(s' \rightarrow e') + \Delta_d)$ **et** $\tau_o(s_i \rightarrow s') + \tau_o(s' \rightarrow v_2) + \tau_o(v_2 \rightarrow e_i) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_{o_i})$ **et** $c_o(s_i \rightarrow e_i) + c_d(s' \rightarrow e') - GlobalPath(s_i, s', s', v_2, e_i, e) \geq 0$ **Alors**
        Si $\sigma^{\star} > GlobalPath(s_i, s', s', v_2, e_i, e')$ **Alors**
          $i^{\star} \leftarrow i$
          $\sigma^{\star} \leftarrow GlobalPath(s_i, s', s', v_2, e_i, e')$
        Fin Si
      Fin Si
    Fin Si
    Si s_i forme une synchronisation temporelle entre le conducteur i le passager d **Alors**
      Si $\tau_d(s' \rightarrow s_i) + \tau_o(s_i \rightarrow v_2) + (\tau_d(v_2 \rightarrow e') \leq (\tau_d(s' \rightarrow e') + \Delta_d)$ **et** $\tau_o(s_i \rightarrow v_2) + \tau_o(v_2 \rightarrow e_i) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_{o_i})$ **et** $c_o(s_i \rightarrow e_i) + c_d(s' \rightarrow e') - GlobalPath(s_i, s', s_i, v_2, e_i, e') \geq 0$ **Alors**
        Si $\sigma^{\star} > GlobalPath(s_i, s', s_i, v_2, e_i, e')$ **Alors**
          $i^{\star} \leftarrow i$
          $\sigma^{\star} \leftarrow GlobalPath(s_i, s', s_i, v_2, e_i, e')$
        Fin Si
      Fin Si
    Fin Si
  Fin Pour
Fin Pour

Algorithme 6 Heuristique de sélection d'offres

Entrée: Demande d, ensemble $N^{\uparrow}(s')$, ensemble $N^{\downarrow}(e')$ et bucket $Bucket_C(v), \forall v \in G$.
Sortie: Le meilleur conducteur $i^{\star}$, trajet-total $\sigma^{\star}$.

1. Déterminer le meilleur conducteur $i_1^{\star}$ dans $N^{\uparrow}(s')$ en utilisant **l'algorithme 4**.
2. Déterminer le meilleur conducteur $i_2^{\star}$ dans $N^{\downarrow}(e')$ en utilisant **l'algorithme 5**.
3. Sélectionner parmi les deux conducteurs $i_1^{\star}$ et $i_2^{\star}$ le conducteur $i^{\star}$ avec le plus petit *trajet-total* $\sigma^{\star} \leftarrow \min(\sigma_1^{\star}, \sigma_2^{\star})$.
4. Déterminer le meilleur lieu de prise en charge r_1 et de dépose r_2 entre le conducteur $i^{\star}$ et la demande d en utilisant la méthode décrite dans la **section 3.3**.

3.5 Expérimentations

Dans cette section, nous allons présenter les performances des deux systèmes séparément : le système *a priori* et le système *a posteriori*.
Dans le système *a priori*, nous avons utilisé des benchmarks de quelques réseaux routiers. Ces derniers correspondent à des villes des États-Unis. Concernant les données, nous avons généré aléatoirement des demandes et des offres de covoiturage selon deux scénarios différents.
En revanche, pour le système a posteriori, nous avons utilisé un vrai réseau routier de la région Lorraine en France ainsi que de vraies données transmises par l'entreprise Covivo [13].

Environnement. Les expériences ont été réalisées sur un processeur 2,4 Ghz Intel Xeon E5620, avec 8 Go de mémoire RAM. Les algorithmes testés pour le système a priori ont été codés en langage C++, tandis que pour le système a posteriori ils ont été codés en C#. Un tas binaire a été utilisé comme structure de données de files d'attente de priorité dans nos différents algorithmes.

3.5.1 Expérimentations partie 1 : Système a priori

Dans le système *a priori* nous avons étudié l'efficacité des deux approches $\mathscr{M}_1$ et $\mathscr{M}_2$ par rapport à la méthode exacte. Nous nous sommes restreints à une demande et une offre fixées sans passer par l'étape de la méthode de sélection de la meilleure offre.

Réseau routier. Nous avons utilisé les graphes disponibles sur DIMACS [14]. Cette plateforme contient des graphes des villes des États-Unis qui peuvent être téléchargés librement. Le tableau 3.1 présente 4 réseaux routiers des villes des État-Unis sur lesquels nous avons testé nos différentes approches. Chaque graphe comprend des distances réelles, des temps de parcours et des coordonnées des nœuds.

Réseaux	Description	# de nœuds	# d'arcs
NY	New York City	264,346	733,846
COL	Colorado	435,666	1,057,066
FLO	Florida	1,070,376	2,712,798
E	Eastern USA	3,598,623	8,778,114

TABLE 3.1 – Instances.

Chaque nœud est considéré comme un emplacement potentiel de prise en charge ou de dépose des passagers. Dans nos expériences, les distances réelles sont considérées comme les coûts de déplacement.

Instances générées. Pour chaque réseau routier, nous générons aléatoirement des offres et des demandes de covoiturage. Plus précisément, nous développons deux scénarios.

Premier scénario. Dans le premier scénario (S_1), nous générons 10 offres $o_i = (s_i, e_i)$. Pour chaque offre o_i générée, nous générons 10 instances. Dans chaque instance, une demande $d_j = (s'_j, e'_j)$ est générée de telle sorte que son emplacement de départ s'_j (respectivement l'emplacement

13. www.covivo.eu
14. http ://www.dis.uniroma1.it/challenge9/download.shtml

e'_j) se trouve dans un rayon $R_1 = r_1 \times c_o(s_i \rightarrow e_i)$ autour de l'emplacement de départ de l'offre s_i (respectivement autour de l'emplacement e_i). Ainsi, 100 instances sont générées dans le scénario 1.

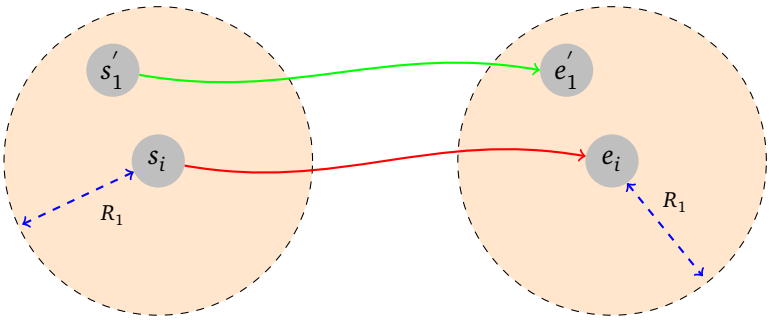


FIGURE 3.9 – Génération d'instances selon le scénario 1. Les emplacements de départ et d'arrivée des passagers sont générés respectivement autour de s_i et e_i. R_1 est le rayon des deux cercles.

Second scénario. Dans le second scénario (S_2), 100 autres instances sont générées. Chaque instance est générée de telle sorte que son emplacement de départ s'_j et d'arrivée e'_j se trouvent dans un rayon $R_2 = r_2 \times c_o(s_i \rightarrow e_i)$ autour de l'emplacement de départ s_i de l'offre o_i (le même raisonnement s'applique autour de l'emplacement d'arrivée e_j).

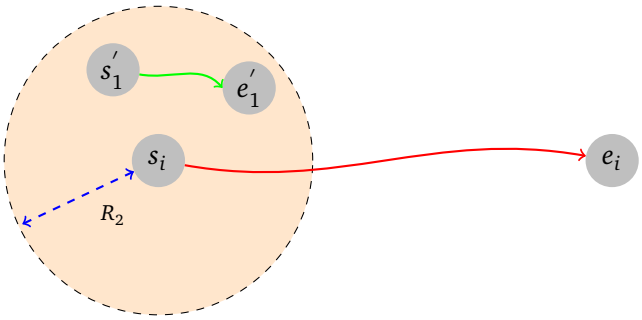


FIGURE 3.10 – Génération d'instances selon le scénario 2. Les emplacements de départ et d'arrivée des passagers sont générés autour d'un seul emplacement s_i (soit le point de départ ou l'arrivée du conducteur). R_2 est le rayon du cerle.

Dans les deux scénarios, les fenêtres de temps des offres et des demandes générées sont fixées à $]-\infty, +\infty[$. Cela signifie que le conducteur et le passager sont prêts à voyager à n'importe quel moment de la journée.
Comme mesures de performance, nous avons déterminé :

- $Match$: Le pourcentage du nombre d'appariements détectés par nos heuristiques et covoiturage classique par rapport à la méthode exacte, c'est à dire, $\sum_{i=0}^{n} \frac{m_i}{n} \cdot 100$ ($m_i = 1$ si l'appariement est détecté par l'heuristique ou le covoiturage classique, 0 sinon, et n correspond au nombre d'appariement detecté par la méthode exacte).
- Gap : L'écart des économies de coûts entre l'appariement détecté par nos heuristiques et la méthode exacte, c'est à dire, $Gap = \frac{Solution_{heuristique} - Solution_{exacte}}{Solution_{exacte}}$.
- $Time$: Le temps d'exécution de l'algorithme en secondes.

Nous supposons que le passager et le conducteur ont la même fonction de coût (c'est à dire $c_o(.,.) = c_d(.,.)$). Le temps de détour maximal est limité à 20% de la durée du trajet direct et le partage de coûts sur le trajet commun se fait équitablement entre le conducteur et le passager (c'est à dire $\varepsilon = 0,5$). Après avoir testé différents rayons possibles, nous avons fixé $r_1 = 0.4$ et $r_2 = 0.3$. Ces deux

valeurs correspondent aux résultats où le nombre d'appariements non détectés est maximal.
Le tableau 3.2 présente les résultats des approches testées : l'approche de covoiturage classique, l'heuristique en utilisant la métrique de partition simple $\mathscr{M}_1$ (notée par HM_1), l'heuristique en utilisant la métrique de partition améliorée $\mathscr{M}_2$ (notée par HM_2) et l'approche énumérative.

Les valeurs dans les deux colonnes Gap et $Time$ correspondent à des valeurs moyennes sur les 100 instances générées pour chaque réseau.

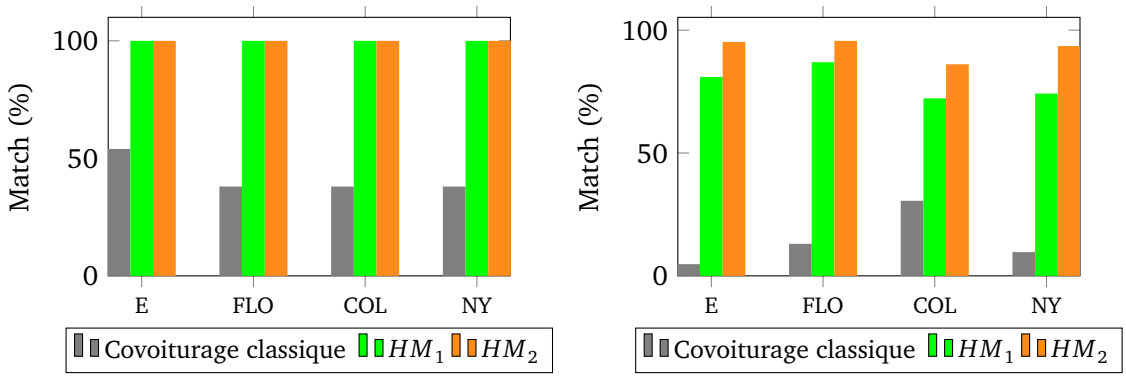
	Réseaux	Covoiturage classique		Heuristique HM_1			Heuristique HM_2			Approche énumérative
		$Match$ (%)	Gap (%)	$Match$ (%)	Gap (%)	$Time$ (s)	$Match$ (%)	Gap (%)	$Time$ (s)	$Time$ (s)
S_1	NY	38	8.14	100	0	0.47	100	0	0.49	450
	COL	38	9.08	100	0	0.78	100	0	0.84	690
	FLO	38	11.01	100	0.02	1.77	100	0.02	1.84	1557.3
	E	54	8.39	100	0	7.32	100	0	7.54	6531.3
S_2	NY	9.67	2.36	74.19	0.83	0.47	93.54	0.07	0.49	421
	COL	30.55	6.18	72.22	0.34	0.8	86.11	0.10	0.86	657
	FLO	13.04	3.43	86.95	0.44	1.79	95.65	0.05	1.85	1642.5
	E	4.76	17.67	80.95	0.01	7.46	95.23	0.03	7.65	6676.2

TABLE 3.2 – Performances des différentes approches.

Les résultats montrent l'efficacité de nos approches proposées :

- Nos heuristiques délivrent des solutions de très hautes qualité. Dans le scénario 1, toutes les demandes ont été appariées pour les deux métriques $\mathscr{M}_1$ et $\mathscr{M}_2$. En revanche, le taux d'appariement dans le covoiturage classique n'excède pas les 54%. Par ailleurs, les deux heuristiques HM_1 et HM_2 obtiennent des solutions optimales pour le réseau E, COL, NY et des solutions quasi-optimales pour le réseau FLO avec un léger $Gap = 0,02\%$.
- Le scénario 2 montre une sensibilité autour de l'heuristique HM_1. Les appariements ne sont pas tous détectés par $\mathscr{M}_1$ dû au partitionnement des nœuds potentiels qui est basé uniquement sur les distances par rapport à (s, s') et (e, e'). Ainsi, les performances $\mathscr{M}_1$ sont détériorées par une détection d'au plus 86,95% des appariements avec un Gap de 0,44%. Concernant l'approche HM_2, le pourcentage d'appariement augmente jusqu'à $95,65\%$ avec un Gap intéressant qui ne dépasse pas les 0,1%. Cela explique la stabilité de la métrique $\mathscr{M}_2$.
  L'approche de covoiturage classique détecte au plus 30% d'appariement avec un Gap qui atteint les 17,67% (voir la figure 3.11).
- Les solutions de nos heuristiques sont également obtenues dans des délais d'exécution plus courts. L'heuristique utilisant la métrique $\mathscr{M}_1$ est légèrement plus rapide que $\mathscr{M}_2$. Ceci peut être expliqué par le fait que l'approche HM_2 nécessite plus de temps en raison de calculs supplémentaires pour la détermination des bornes inférieurs, mais il reste de l'ordre des millisecondes.

Concernant les résultats du scénario 2 sur les instances COL et E, $\mathscr{M}_1$ est aussi efficace que $\mathscr{M}_2$ en termes de Gap. Cela est dû au fait que le Gap et les économies de coûts moyens sont calculés sur l'ensemble des appariements réussis alors que $\mathscr{M}_2$ détecte plus d'appariement que $\mathscr{M}_1$.
Pour avoir une meilleure idée sur la performance de chaque métrique, nous comparons les deux approches HM_1 et HM_2 sur des instances où un appariement est réussi seulement par ces deux dernières. Ces instances sont notées par I.



(a) Scénario 1 (b) Scénario 2

FIGURE 3.11 – Visualisation du pourcentage d'appariement (*Match*) pour les deux scénarios.

Les résultats sont présentés dans le tableau 3.3. La colonne $B_{\mathscr{M}_1}$ présente le pourcentage du nombre d'appariements dans le cas où une meilleure solution est obtenue seulement par HM_1. La colonne $B_{\mathscr{M}_2}$ présente le pourcentage du nombre d'appariements dans le cas où une meilleure solution est obtenue seulement par HM_2.

Scénarios	Réseaux	HM_1		HM_2	
		$B_{\mathscr{M}_1}$ (%)	Gap (%)	$B_{\mathscr{M}_2}$ (%)	Gap (%)
S_1	NY	0	0	0	0
	COL	0	0	0	0
	FLO	0	0,02	0	0,02
	E	0	0	0	0
S_2	NY	0	0,84	5,55	0,02
	COL	0	0,02	15,38	0
	FLO	0	0,44	25	0,03
	E	0	0,01	34,78	0

TABLE 3.3 – Performance des métriques HM_1 et HM_2 sur les instances I.

D'après le tableau 3.3, nous observons que HM_2 surpasse HM_1, en l'occurrence dans le scénario 2. En effet, pour FLO et NY, le Gap de l'approche HM_1 est respectivement de $0,44\%$ et $0,84\%$. En revanche pour HM_2, le Gap est respectivement de $0,03\%$ et $0,02\%$.

3.5.2 Expérimentation partie 2 : Système a posteriori

Dans cette partie, nous allons nous focaliser sur le système *a posteriori*. Dans un premier temps, nous fixons nous même les offres avec lesquelles les demandes seront appariées sans passer par la méthode de sélection d'offres. Ceci nous permet de tester l'efficacité de nos approches **RBM** et **PCCM** par rapport à la méthode exacte **ME** de sélection de lieux intermédiaires ainsi que le covoiturage classique, noté **CR**. Dans un second temps, nous testons notre heuristique de sélection d'offres par rapport à la méthode exacte de sélection d'offres.

Données réelles. Dans cette deuxième partie, nous avons utilisé des données réelles fournies par une entreprise locale de covoiturage [15]. Ces données correspondent aux employées de la région Lorraine voyageant entre leur domicile et leur lieu de travail. Ces données sont composées de 756 offres et 757 demandes pour un voyage domicile-travail. En ce qui concerne les demandes, chaque passager est prêt à utiliser son véhicule privé pour atteindre l'emplacement intermédiaire de prise en charge à partir de son point de départ ainsi que sa destination depuis l'emplacement intermédiaire de dépose. L'ensemble des offres et des demandes est filtré pour que l'on ne puisse pas trouver une offre et une demande qui ont à la fois le même lieu départ et la même destination. La fenêtre de temps de départ ainsi que le temps de détour de chaque trajet sont fixés comme suit :

- L'heure de départ au plus tôt et l'heure de départ au plus tard sont respectivement fixés à 07h30 et à 08h00.
- Le temps de détour du conducteur (passager) est fixé au plus à 20% de la durée de son trajet direct.

Dans la figure 3.12 nous projetons les offres de covoiturage. Les points bleus et rouges correspondent respectivement aux lieux de départ et de destination des conducteurs.

Réseau routier. Les applications réelles de routage ont besoin de cartes géographiques, de sorte que les algorithmes de routage puissent y être appliqués. Il existe plusieurs données cartographiques exploitables, telles que google maps [16], qui est l'une des plus connue, les cartes Microsoft Bing [17], les cartes de Nokia [18] ou encore OpenStreetMap [19] (OSM).
OpenStreetMap est un projet libre de constitution d'une base de données géographiques mondiales. L'ajout et la correction des données dans la base sont réalisés de manière collaborative par des contributeurs volontaires. Nous avons choisi OSM pour tester notre application. Les parties de cartes OSM téléchargeables sont disponibles sur le site de Geofabrik [20] par exemple. Nous avons utilisé OsmSharp [21] pour le traitement de données cartographiques. Il utilise le langage de programmation C# développé par Microsoft. Nous avons également utilisé ses fonctionnalités de traitement de données OSM ainsi que sa librairie de routage afin de tester nos algorithmes. Notre réseau routier correspond à celui de la région Lorraine qui est constitué de 797 830 nœuds et 2 394 002 d'arcs. Chaque nœud de ce réseau est considéré comme un emplacement intermédiaire potentiel dans lequel le conducteur et le passager peuvent se rencontrer. Plus spécifiquement, un noeud du réseau (autre que les lieux de départ des passagers et des conducteurs) ayant un seul prédécesseur ne pourrait jamais correspondre à un lieu intermédiaire de rencontre. De même, un noeud du réseau (autre que les lieux d'arrivée des passagers et des conducteurs) ayant un seul successeur ne pourrait jamais correspondre à un lieu intermédiaire de séparation. Cela permet de filtrer un grand ensemble de nœuds du réseau.

Première partie : Offre fixée à l'avance

Le tableau 3.4 représente les réductions de coûts, le nombre d'appariements ainsi que le temps de calcul de nos trois approches. Deux scénarios ont été testés. Dans le premier scénario, le temps de détour du conducteur à été limité à 20% tandis que dans le deuxième scénario le temps de détour est limité à 10%. Dans chaque scénario, plusieurs instances sont générées comme suit : nous parcourons toutes les demandes. Pour chaque demande parcourue, nous choisissons au hasard 10 offres et ainsi 7570 instances sont générées. Les mesures de performance restent les mêmes que celles décrites pré-

15. http ://www.covivo.fr
16. https ://maps.google.com/
17. http ://www.bing.com/maps/
18. http ://m.here.com
19. http ://www.openstreetmap.org
20. http ://download.geofabrik.de/
21. http ://www.osmsharp.com/

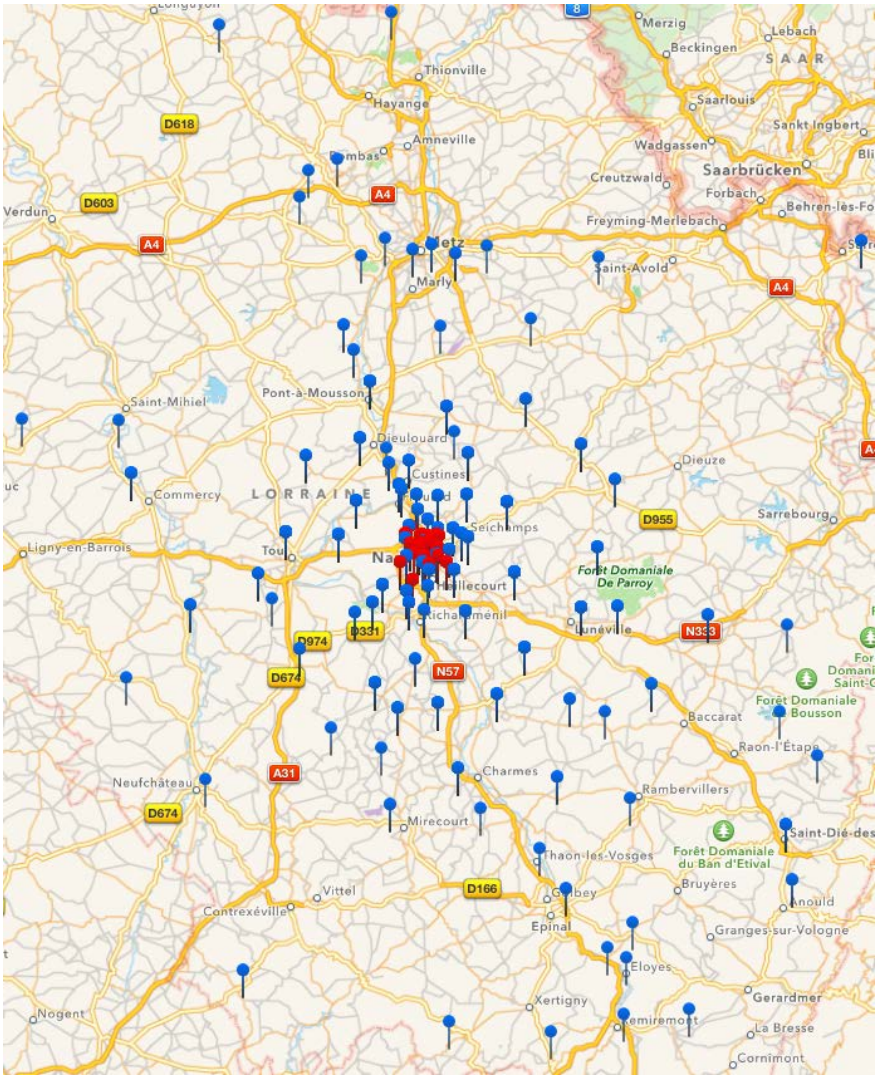


FIGURE 3.12 – Les offres de covoiturage projetées sur une carte géographique.

cédemment. La ligne ($|C|$) représente le nombre moyen de noeuds contenu dans la classe potentielle de lieux intermédiaires (C) sur les 7570 instances.

Approches	Temps de détour ≤ 0.2			Temps de détour ≤ 0.1		
	Gap (%)	*Match* (%)	*Time* (s)	*Gap* (%)	*Match* (%)	*Time* (s)
ME	-	-	844	-	-	620
RBM	0	100	1.78	0.4	94	1.22
PCCM	0	100	2.10	0.2	97	1.58
CR	27	55.7	0.92	34	15	0.83
$\|C\|$	742			450		

TABLE 3.4 – Performances de nos approches avec une offre fixée à l'avance.

Comme nous pouvons le constater, les heuristiques proposées (**RBM** et **PCCM**) détectent tous les appariements et obtiennent des solutions exactes pour les instances où le temps de détour est inférieur à 20%. En revanche, lorsque l'heure de détour est inférieur à 10%, le pourcentage d'appariement ($Match$) de l'approche **RBM** et **PCCM** diminue respectivement à 94% et 97%.
D'autre part, l'approche **CR** détecte au plus 55,7% lorsque le temps de détour est moins de 20% et diminue considérablement à 15% pour un temps détour inférieur à 10 %.
Le Gap de l'approche **CR** peut être considéré comme les économies supplémentaires générées en introduisant des emplacements intermédiaires de prise en charge et de dépose dans le covoiturage.
Le temps d'exécution de l'approche **RBM** est plus rapide que celui de l'approche **PCCM**, cela est dû à l'algorithme de Dijkstra un-vers-plusieurs utilisé dans **PCCM** comparé au bidirectionnelle de l'approche **RBM**.

Deuxième partie : Offre non fixée
Dans ce qui suit, nous évaluons l'heuristique de sélection d'offres. Cela nous permet de savoir si en pratique l'offre choisie en se basant sur un seul emplacement intermédiaire (soit la prise en charge ou le lieu de dépose) est généralement la même offre sélectionnée trouvée en utilisant les deux emplacements intermédiaires (voir le tableau 3.5). La colonne "Time" représente le temps d'exécution total de l'algorithme en secondes, alors que la ligne "Same offer (SO)" représente le taux de réussite dans lequel l'offre sélectionnée par l'heuristique (étape 3 de l'**algorithm 6**) est la même que celle sélectionnée par la méthode exacte (**ME**).

Approches	Temps de détour ≤ 0.2			Temps de détour ≤ 0.1		
	Gap (%)	SO (%)	$Time$ (s)	Gap (%)	SO (%)	$Time$ (s)
ME	-	-	9409	-	-	7803
RBM	0.7	91	3.79	2.1	83	2.64
PCCM	0.3	91	5.22	1.4	83	3.87
$\sum_{v \in C} \lvert Bucket_C(v) \rvert$	113836			84742		

TABLE 3.5 – Performance du processus global.

D'après le tableau 3.5, 83% (resp. 91% pour un temps de détour $\leq 0,2$) des offres sélectionnées en utilisant un emplacement intermédiaire sont les mêmes que celles sélectionnées en utilisant les deux emplacements intermédiaires. Lorsque le temps détour est inférieur à 20%, l'appariement est déterminé approximativement en 4 secondes pour l'approche **RBM** et en 5 secondes pour **PCCM**. En revanche, la méthode exacte atteint les 9409 secondes. Le Gap entre les deux heuristiques ne dépasse pas 2,1% dans les deux scénarios.

Afin d'évaluer le temps nécessaire d'ajout et de suppression d'une offre, nous avons fixé trois offres o_1, o_2 et o_3 composées respectivement de 99, 239 et 549 buckets. Ces dernières sont ajoutées/supprimées du système en fonction du nombre d'offres dans les buckets.

Comme le montre les résultats du Tableau 3.6, le temps d'ajout d'une offre o_i est indépendant du nombre d'offres déjà dans le système. Le temps de calcul est plutôt consacré dans la détermination des deux ensembles $M^{\uparrow}(s_i)$ et $M^{\downarrow}(e_i)$. En revanche, la suppression de l'offre nécessite le parcours de ces buckets. Toutefois, le temps d'exécution pour l'ajout et la suppression d'une offre reste à l'échelle de la seconde.

Offres	# buckets	# offres dans les buckets	Ajout [s]	Suppression [s]
o_1	99	100	0.03	0.032
		250	0.03	0.041
		550	0.03	0.053
o_2	239	100	0.05	0.052
		250	0.05	0.062
		550	0.05	0.074
o_3	549	100	0.08	0.083
		250	0.08	0.093
		550	0.08	0.1

TABLE 3.6 – Ajout et suppression d'une offre de covoiturage aller-retour.

La figure 3.13 donne une meilleure visualisation du temps d'ajout et de la suppression d'une offre.

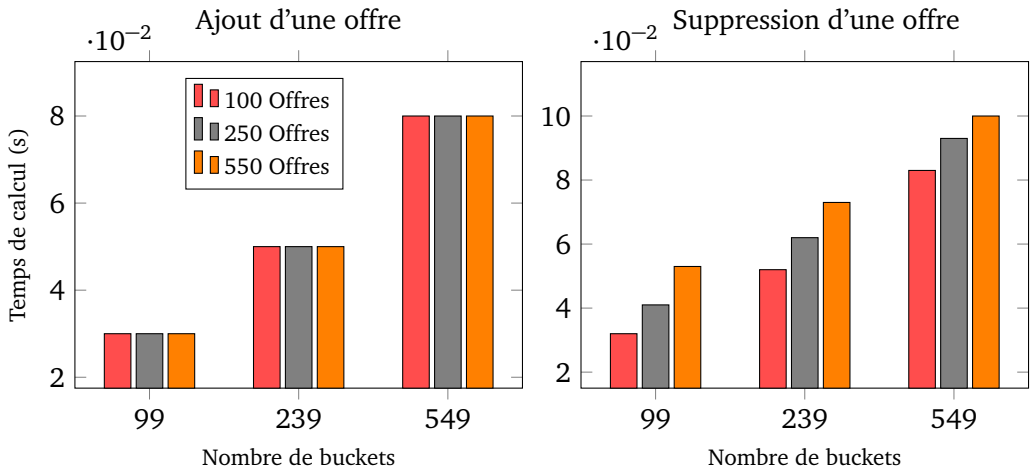


FIGURE 3.13 – Ajout et suppression d'une offre en fonction du nombre d'offres et de buckets dans le système.

La figure 3.14 projette le gain sur les temps de calcul que nous économisons si les deux espaces de recherche sont guidés. Le gain observé n'est pas très élevé mais reste indispensable dans un contexte en temps réel. Ce gain est dû au fait que les deux algorithmes Dijkstra utilisés dans notre approche (l'ajout d'une offre de covoiturage décrite dans la section 4.4.1) sont guidés par des fonctions heuristiques (algorithme $A^\star$).

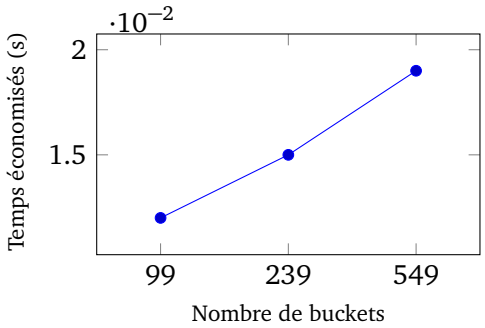


FIGURE 3.14 – Temps de calcul économisé sur l'opération d'ajout en guidant les deux espaces de recherche $N^{\uparrow}(s_i)$ et $N^{\downarrow}(e_i)$.

3.6 Conclusion du chapitre

Dans cette chapitre, nous avons présenté un système de covoiturage dans lequel les emplacements de prise en charge et de dépose des passagers sont flexibles. Nous avons présenté deux types de variantes pour ce système. Le système *a priori* et le système *a posteriori*. Le système *a posteriori* est plus utilisé car généralement les utilisateurs ne fixent pas le taux de partage de coûts à l'avance tant que le système lui-même leur garantit des économies de coûts en optant pour le covoiturage. Pour résoudre ce problème, nous avons proposé différentes méthodes pour les deux systèmes.

4

Covoiturage aller-retour avec stations relais

Sommaire

Dans ce chapitre, nous étudions l'intérêt d'introduire des stations relais dans le problème de covoiturage aller-retour. Dans le système classique de covoiturage aller-retour, l'emplacement de prise en charge et de dépose du passager ne diffère pas de son origine et de sa destination. Ce système est non flexible et non équilibré car la totalité du détour repose sur le conducteur. Pour ce faire, nous considérons l'emplacement de la prise en charge du passager comme un emplacement flexible.
L'emplacement optimal de ce dernier sera déterminé de manière à ce que le coût du trajet total aller-retour soit minimisé. Ces emplacements de rencontre correspondent aux stations relais dans lesquelles les passagers trouvent des véhicules de covoiturage. Contrairement au système classique aller-retour, l'introduction de stations relais dans le problème de covoiturage aller-retour crée une dépendance entre le trajet aller et le trajet retour. Cette dépendance réside dans le choix de la station relais qui doit être déterminée de manière à réduire au maximum le coût du trajet total aller-retour. Dans cet arrangement, le passager est censé se déplacer avec son propre véhicule jusqu'à la station relais. Cette dernière servira de lieu de rencontre entre le passager et le conducteur ainsi qu'un emplacement de stationnement pour garer son véhicule privé en attendant son trajet retour. Lors de ce dernier, le passager doit être déposé à la station relais de départ afin qu'il puisse récupérer son véhicule pour terminer son trajet. Nous présentons des algorithmes efficaces pour résoudre ce problème, nous les avons validé sur un vrai réseau routier et sur des données réelles fournies par l'entreprise de covoiturage Covivo. Nos résultats numériques montrent l'efficacité de notre approche comparée

au système classique de covoiturage aller-retour. Plus précisément, une amélioration significative de la réduction des coûts des trajets ainsi que du nombre d'appariement réussi dans le système.

4.1 Introduction

Bien que le covoiturage présente beaucoup d'avantages, la plupart des utilisateurs sont réticents pour participer comme passager à un tel service. Cela concerne particulièrement les navetteurs qui ne sont pas prêts à laisser leur voiture personnelle pour covoiturer avec d'autres conducteurs si leur voyage retour n'est pas assuré par le service.
Très peu de recherches se sont concentrées sur le problème de covoiturage aller-retour. L'étude présentée dans [Agatz et al., 2011] traite le problème de covoiturage aller-retour dans lequel chaque utilisateur peut être conducteur ou passager avec comme objectif la minimisation des kilomètres parcourus par l'ensemble des véhicules. La décision consiste à attribuer un rôle à chaque utilisateur dans le but de former des appariements optimaux entre des conducteurs et des passagers. Dans le cas où l'utilisateur est identifié comme étant un passager, leur système assure un autre appariement pour le trajet retour. Dans [Baldacci et al., 2004], les auteurs étudient le problème de covoiturage aller-retour dans lequel ils considèrent que le trajet aller est indépendant de celui du retour et que ces derniers doivent être résolus séparément. Les deux trajets n'étant pas corrélés, les auteurs ont résolu une seule variante qui correspond au trajet aller. Dans leur modèle, les lieux de départ correspondent aux domiciles des employés, tandis que le lieu de travail est fixé à un seul endroit. Or, dans des grandes zones d'activités ou des zones industrielles, les lieux de travail sont distincts et espacés les uns des autres.

[Huguet et al., 2013] propose une approche pour le calcul d'un plus court chemin multimodal aller-retour "2-way multi-modal shortest path problem". Leur approche consiste à déterminer deux itinéraires multimodaux entre un point origine et un point destination, par exemple un trajet aller domicile-travail et un trajet retour travail-domicile, de telle sorte que le coût de l'itinéraire aller-retour soit minimal. Une caractéristique similaire à notre problème est que les deux itinéraires doivent se «rencontrer» en un point du réseau. Leur approche est basée sur le calcul de quatre algorithmes de plus court chemins multimodaux de type Dijkstra, exécutés parallèlement depuis l'origine et la destination du passager. Deux d'entre eux détermineront le trajet aller et les deux autres détermineront le trajet retour. Les deux algorithmes de Dijkstra sortants de la source sont limités à l'utilisation d'un véhicule personnel alors que les autres sont limités à l'utilisation des transports en commun et la marche à pied. A chaque étape, le sommet du plus petit coût pour l'un de ces 4 algorithmes est sélectionné, marqué, et ses successeurs sont mis à jour. Lorsqu'un nœud est marqué par les 4 algorithmes, une borne supérieure de la solution optimale est atteinte. Le processus se répète jusqu'à ce qu'un nœud de plus petit coût ait une valuation supérieure ou égale à la borne supérieure. Des améliorations de cette méthode ont été également proposées. Ces améliorations utilisent un calcul de bornes inférieures s'appuyant sur les coûts minimums pour une connexion de plusieurs algorithmes ou des techniques d'accélération dédiées pour le cas dépendant du temps [Kirchler et al., 2011]. Leur approche est très efficace mais elle n'est pas extensible pour intégrer le covoiturage comme un mode de transport.
D'autre part, dans les systèmes existants de covoiturage aller-retour, le lieu de prise en charge du passager ne diffère pas de son lieu de départ. Cependant, dans certaines situations le passager peut accepter de se déplacer avec sa voiture personnelle à une station relais que l'on peut considérer comme un nouvel emplacement de prise en charge. Ce dernier sera plus ou moins proche de son emplacement initial de départ, là où il sera pris en charge par le conducteur et enfin déposé à sa des-

tination. En plus du trajet-aller, notre système de covoiturage doit également garantir au passager l'existence d'un nouvel appariement pour son trajet retour passant par la station relais. Il s'agit d'un appariement avec un autre/même conducteur qui doit récupérer le passager depuis sa destination vers la station relais, là où sa voiture a été laissée pendant son trajet aller. L'objectif de ce chapitre est double :

(i) L'objectif principal est d'augmenter le nombre d'appariements entre les passagers et les conducteurs pour le système de covoiturage aller-retour. En effet, si un passager accepte de se déplacer avec sa propre voiture à une station relais proche de son emplacement d'origine, le conducteur fera moins de détour pour prendre en charge le passager. Cela permet également de réduire le coût total du trajet.

(ii) Le deuxième objectif est d'assurer pour le passager son trajet retour en passant par la station relais où il a laissé sa voiture personnelle. En effet, le passager ne sera pas prêt de laisser sa voiture dans une station relais pour participer à un tel service si son trajet retour passant par cette station relais n'est pas assuré par le système.

A notre connaissance, notre travail est le premier à introduire le problème de covoiturage aller-retour avec stations relais. Ce problème consiste à minimiser le coût total des deux trajets aller-retour. Nous considérons également un cadre pratique en exploitant le réseau routier de la région Lorraine avec une validation sur de vraies données.

Le reste du chapitre est structuré comme suit. La section 2 décrit notre modèle de covoiturage aller-retour. La section 3 présente les détails algorithmiques de notre approche. La section 4 présente une extension du modèle. La section 5 présente l'analyse expérimentale détaillée de nos algorithmes. Enfin, une conclusion et des perspectives sont présentées dans la section 6.

4.2 Description de l'approche

Le graphe utilisé dans cette approche est le même que celui décrit dans la partie 3.2. Une offre de covoiturage i est représentée par o_i=($s_i, e_i, [t_i^{\min}, t_i^{\max}], \Delta_i$), où s_i est la point de départ, e_i est le point d'arrivée, $[t_i^{\min}, t_i^{\max}]$ la fenêtre de temps de départ et Δ_i est le temps de détour. Le temps de détour représente la durée maximale que le conducteur peut tolérer en plus de la durée de son trajet direct. Lorsqu'un conducteur souhaite effectuer un déplacement en aller-retour, deux offres sont générées, la première pour le trajet aller et la seconde pour le trajet retour. En revanche pour une demande de covoiturage, le trajet aller et retour sont regroupés dans une seule requête. Cette dernière est représentée par d=($s', e', [t_{\min}^{aller}, t_{\max}^{aller}], [t_{\min}^{retour}, t_{\max}^{retour}], \Delta_d^{aller}, \Delta_d^{retour}$), où s' est le point de départ, e' le point d'arrivée, $[t_{\min}^{aller}, t_{\max}^{aller}]$ et $[t_{\min}^{retour}, t_{\max}^{retour}]$ sont les fenêtres de temps respectives de départ pour les trajets aller et retour. Le temps de détour pour le trajet aller est indépendant de celui du trajet retour qui sont représentés respectivement par Δ_d^{aller} et Δ_d^{retour}. De même, un arc (i, j) peut être emprunté soit par un conducteur soit par un passager. Dans le cas où l'arc est emprunté par un passager, la fonction coût sera indexée par d sinon par o pour un conducteur.
Dans [Agatz et al., 2011], les auteurs considèrent une contrainte sur une réduction de coût $cs(o_i, o_j, d)$ comme une condition nécessaire pour un *appariement raisonnable* entre deux offres et une demande de covoiturage. La réduction de coût du trajet est définie comme une différence entre le coût du trajet du conducteur i (resp. conducteur j) incluant son coût de déviation pour servir le passager sur son trajet aller (resp. trajet retour) et la somme des coûts des quatre trajets (passager aller, passager retour, conducteur i et conducteur j) si chacun d'eux voyageait seul, c'est à dire :

$$cs(o_i, o_j, d) = c_o(s_i \to e_i) + c_d(s' \to e') + c_o(s_j \to e_j) + c_d(e' \to s') - \big(c_o(s_i \to s') + c_o(s' \to e') + c_o(e' \to e_i) + c_o(s_j \to e') + c_o(e' \to s') + c_o(s' \to e_j)\big) \quad (4.1)$$

Ainsi, un appariement est dit raisonnable si seulement si le coût d'un *Trajet en covoiturage* est inférieur au coût des deux trajets séparés, c'est à dire $cs(o_i, o_j, d) > 0$. Si le coût du *Trajet en covoiturage* est plus élevé que le coût cumulé des quatre trajets individuels, l'appariement ne sera pas pris en considération (voir la figure 4.1).

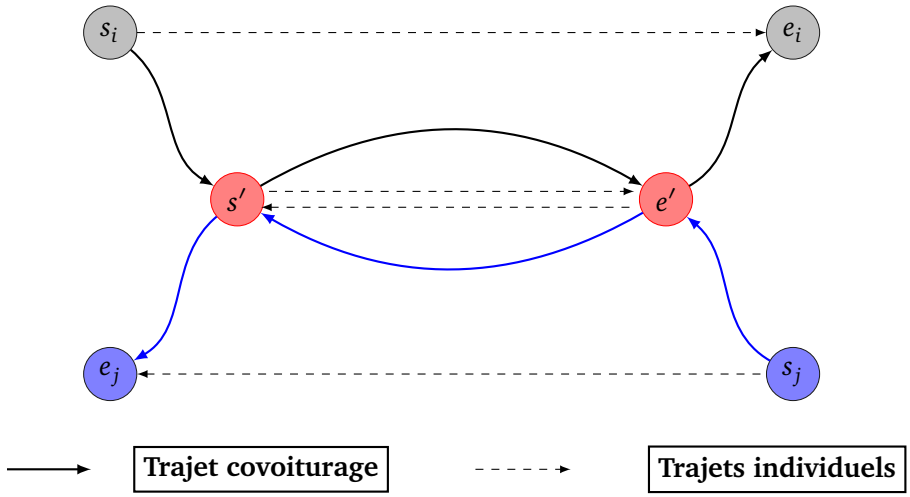


FIGURE 4.1 – Exemple d'un appariement entre un passager et un conducteur.

Afin d'augmenter les chances de succès sur le nombre appariements, le passager peut accepter un emplacement de pris en charge v autre que son lieu de départ. Plus précisément, le passager se déplace avec sa voiture personnelle de son emplacement de départ s' à la station relais v.
Il stationne sa voiture à la station relais v afin qu'il puisse partager un sous-trajet avec un conducteur i depuis la station relais v jusqu'à sa destination finale e'. Tandis que le conducteur i se déplace de son emplacement de départ s_i jusqu'à la station v, il prend en charge le passager pour voyager ensemble jusqu'à l'emplacement de dépose du passager e', et enfin le conducteur continue son chemin jusqu'à sa destination finale e_i. Le système doit également garantir au passager l'existence d'un deuxième appariement avec un conducteur j pour atteindre la station relais v depuis e' pour son trajet retour. Les chemins du passager d et des deux conducteurs i et j sont représentés dans la figure 4.2.

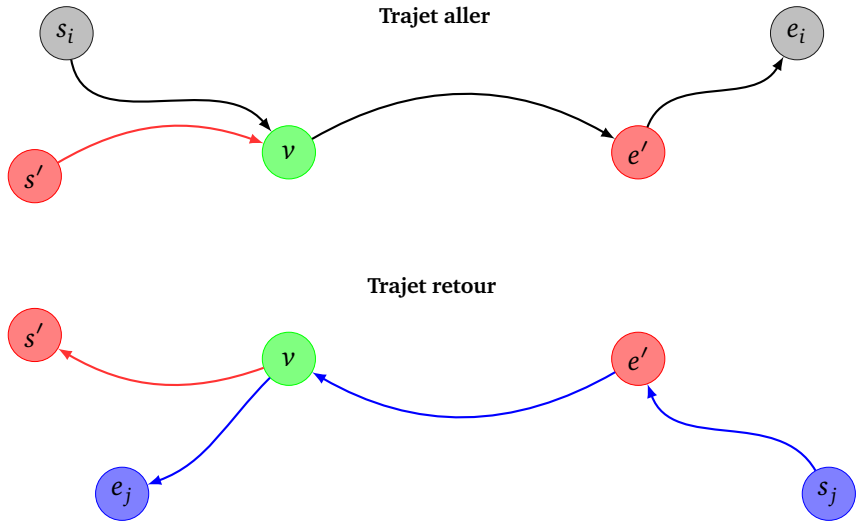


FIGURE 4.2 – Covoiturage aller-retour avec v comme station relais.

4.2.1 Contraintes d'appariement

Le covoiturage nécessite notamment une parfaite synchronisation des participants. D'autres points relèvent d'ententes négociées, à l'instar des détours dans le parcours où des affinités. Dans notre approche, nous considérons deux contraintes d'appariement, à savoir : la contrainte de synchronisation temporelle (ou de timing) (voir la Définition 4 dans la section 3.2.2) et la contrainte de coût/temps du trajet.

Definition. 9 (Appariement raisonnable pour le trajet aller ayant v comme station relais) *Une demande d et une offre o_i forment un appariement raisonnable pour le trajet aller avec v comme station relais si et seulement si d et o_i forment une synchronisation temporelle à la station relais v pour le trajet aller et les contraintes suivantes sont satisfaites :*

$$c_o(s_i \to e_i) + c_d(s' \to e') - \big(c_o(s_i \to v) + c_d(s' \to v) + c_o(v \to e') + c_o(e' \to e_i)\big) \geq 0 \tag{4.2}$$

$$\tau_o(s_i \to v) + \tau_o(v \to e') + \tau_o(e' \to e_i) \leq \tau_o(s_i \to e_i) + \Delta_i \tag{4.3}$$

$$\tau_d(s' \to v) + \tau_o(v \to e') \leq \tau_d(s' \to e') + \Delta_d^{aller} \tag{4.4}$$

La contrainte (4.2) assure l'attractivité du coût encouru en faisant du covoiturage par rapport aux coûts cumulés du conducteur et du passager si chacun voyage de son côté. Les contraintes (4.3) et (4.4) correspondent respectivement au temps de détour du passager et du conducteur pour le trajet aller. Le terme $\tau_o(s_i \to e_i) + \Delta_i$ dans (4.3) (resp. $\tau_d(s' \to e') + \Delta_d^{aller}$ dans (4.4)) permet de limiter la quantité de temps que le conducteur (resp. passager) passe dans son trajet aller. De même, le temps de service pour la prise en charge et de dépose des passagers n'est pas significatif et peut être considéré comme instantané. La prise en compte de ce temps de service ne complexifie pas notre problème.

Definition. 10 (Appariement raisonnable pour le trajet retour avec v comme station relais) *Une demande d et une offre o_j forment un appariement raisonnable pour le trajet retour avec v comme station relais si et seulement si d et o_j forment une synchronisation temporelle à la station v pour le trajet retour et si les contraintes suivantes sont satisfaites :*

$$c_o(s_j \to e_j) + c_d(e' \to s') - (c_o(s_j \to e') + c_o(e' \to v) + c_d(v \to s') + c_o(v \to e_j)) \geq 0 \tag{4.5}$$

$$\tau_o(s_j \to e') + \tau_o(e' \to v) + \tau_o(v \to e_j) \leq (\tau_o(s_j \to e_j) + \Delta_j) \tag{4.6}$$

$$\tau_o(e' \to v) + \tau_d(v \to s') \leq (\tau_d(e' \to s') + \Delta_d^{retour}) \tag{4.7}$$

La contrainte (4.5) assure une réduction des coûts engendrés par le passager d et le conducteur j pour le trajet retour. En revanche, les contraintes (4.6) et (4.7) correspondent respectivement au temps de détour pour le conducteur et le passager.

Lemme. 3 *Une demande d et deux offres o_i et o_j forment un appariement raisonnable pour le trajet aller-retour si et seulement si il existe une station relais v telle que :*

(i) d et o_i forment un appariement raisonnable pour le trajet aller avec v comme station relais.

(ii) d et o_j forment un appariement raisonnable pour le trajet retour avec v comme station relais.

4.2.2 Objectif du système de covoiturage aller-retour avec une station relais

Dans ce qui suit, nous utiliserons le terme *trajet-aller-retour* $\langle d, i, j, v\rangle$ pour décrire le trajet aller-retour du passager d avec le conducteur i pour le trajet aller et le conducteur j pour le trajet retour passant par la station relais v.

Un plus court *trajet-aller-retour* $\langle d, i, j, v\rangle$ est le *trajet-aller-retour* avec un coût minimal dénoté par $c(\langle d, i, j, v\rangle)$, tel que :

$$c(\langle d, i, j, v\rangle) = c(\langle d, i, v\rangle)_{aller} + c(\langle d, j, v\rangle)_{retour} \tag{4.8}$$

où

$$c(\langle d, i, v\rangle)_{aller} = c_o(s_i \to v) + c_d(s' \to v) + c_o(v \to e') + c_o(e' \to e_i) \tag{4.9}$$

et

$$c(\langle d, j, v\rangle)_{retour} = c_o(s_j \to e') + c_o(e' \to v) + c_o(v \to e_j) + c_d(v \to s') \tag{4.10}$$

Ainsi, pour une demande donnée d, notre objectif est de déterminer une offre o_i pour le trajet aller, une station relais v et une autre offre o_j pour le trajet retour. Le coût du *trajet-aller-retour* doit être minimisé tout en respectant les contraintes d'appariement.

4.3 Algorithme de résolution

Dans cette section, nous expliquons les détails algorithmiques de nos approches de résolution (approches polynomiales). Il est à noter que le système est exécuté lorsqu'une demande de covoiturage entre dans le système. En revanche, lorsqu'un conducteur propose une offre de covoiturage, certaines informations sont stockées.

4.3.1 Ajout d'une offre

La procédure d'ajout d'une offre est assez similaire à celle décrite dans la section 3.4.1.

Cette dernière consiste à sélectionner les stations relais possibles pour lesquelles le conducteur peut passer sans enfreindre la contrainte du temps de détour. Par conséquent, les informations stockées dans le bucket $Bucket_C(v)$ ne sont pas tout à fait les mêmes. En effet, pour un nœud donné v marqué par les deux espaces de recherche $M^{\uparrow}(s_i)$ et $M^{\downarrow}(e_i)$, nous devons uniquement stocker les informations nécessaires si ce dernier est choisi comme une station relais. Le coût $c^{\uparrow}_{s_i}$ et le temps nécessaire $\tau^{\uparrow}_{s_i}$ pour rejoindre la station v depuis son emplacement de départ s_i seront stockés. Ainsi, les informations ajoutées au bucket $Bucket_C(v)$ sont les suivantes :

$$Bucket_C(v) := \{(i, c^{\uparrow}_{s_i}, \tau^{\uparrow}_{s_i}) \mid (\tau^{\uparrow}_{s_i} + \tau^{\downarrow}_{e_i}) \leq (\tau_o(s_i \rightarrow e_i) + \Delta_i)\}. \tag{4.11}$$

La durée et le coût du trajet restant (notés par $\tau^{\downarrow}_{e_i}$ et $c^{\downarrow}_{e_i}$) pour que le conducteur puisse atteindre sa destination ne sont pas stockés dans le bucket $Bucket_C(v)$. Ces derniers sont stockés uniquement dans l'ensemble $M^{\downarrow}(e_i)$. Autrement dit, lors d'un parcours d'un $Bucket_C(v)$, les informations requises sont uniquement $\tau^{\uparrow}_{s_i}$ et $c^{\uparrow}_{s_i}$. Ce n'était pas le cas dans la procédure d'ajout décrite dans la section 3.4.1 car le lieu intermédiaire v pouvait convenir soit à un lieu de rencontre, soit à un lieu de séparation (d'où la nécessité dans la procédure d'ajout décrite dans la section 3.4.1 d'avoir les deux informations supplémentaires $\tau^{\downarrow}_{e_i}$ et $c^{\downarrow}_{e_i}$ dans le même bucket $Bucket_C(v)$). De même, pour prendre en considération les conducteurs ayant déjà entamé leur voyage, on doit actualiser leur position à chaque unité de temps. De même, nous considérons seulement les conducteurs qui n'ont pas encore entamé leur voyage lors d'une arrivée d'une demande de covoiturage.

4.3.2 Ajout d'une demande

Pour une demande donnée d, l'objectif de notre procédure est de sélectionner le meilleur *appariement aller-retour* en parcourant les *buckets* des nœuds marqués par les deux recherches aller et retour. Plus précisément, nous déterminons une classe de stations relais potentielles C, où la contrainte du temps de détour du passager est simultanément respectée dans le trajet aller et le trajet retour.
L'algorithme 7 permet de déterminer cette classe C. Les ensembles $N^{\uparrow}(s')$, $N^{\downarrow}(e')$, $N^{\downarrow}(s')$ et $N^{\uparrow}(e')$ sont définit de la même façon que dans la section 3.3.1.

Algorithme 7 Stations relais potentielles pour le trajet aller-retour

Entrée: Graphe $G(V,A)$, demande d.

Sortie: Classe de stations relais potentielles pour le trajet aller-retour C.

1: **Initialisation** : $C_{aller} \leftarrow \emptyset$, $C_{retour} \leftarrow \emptyset$.
2: Déterminer $N^{\uparrow}(s')$, en utilisant l'algorithme de Dijkstra un-vers-plusieurs (forward one-to-all Dijkstra algorithm) depuis s' avec un coût passager dont le rayon de recherche est limité à $(\tau_d(s' \rightarrow e') + \Delta^{aller}_d)$.
3: Déterminer $N^{\downarrow}(e')$, en utilisant l'algorithme Dijkstra-inverse un-vers-plusieurs (backward one-to-all Dijkstra algorithm) depuis e' avec un coût conducteur dont le rayon de recherche est limité à $(\tau_d(s' \rightarrow e') + \Delta^{aller}_d)$. Pour chaque station relais v marquée par la recherche et qui satisfait la contrainte (4.4), $C_{aller} \leftarrow C_{aller} \cup \{v\}$.
4: Déterminer $N^{\downarrow}(s')$, en utilisant l'algorithme Dijkstra-inverse un-vers-plusieurs (backward one-to-all Dijkstra algorithm) depuis s' avec un coût passager dont la recherche est limitée à $(\tau_d(e' \rightarrow s') + \Delta^{retour}_d)$.
5: Détermination de $N^{\uparrow}(e')$, en utilisant l'algorithme de Dijkstra un-vers-plusieurs (forward one-to-all Dijkstra algorithm) depuis e' avec un coût passager dont la recherche est limitée à $(\tau_d(e' \rightarrow s') + \Delta^{aller}_d)$. Pour chaque station relais v marquée par la recherche et qui satisfait la contrainte (4.7), $C_{retour} \leftarrow C_{retour} \cup \{v\}$.
6: $C \leftarrow C_{aller} \cap C_{retour}$.

Les étapes 2 et 3 de l'algorithme 7 permettent de déterminer toutes les stations relais v qui respectent la contrainte du temps de détour pour le trajet aller (4.4). Le même raisonnement s'applique dans les étapes 4 et 5 avec la contrainte du temps de détour pour le trajet retour (4.7). Enfin à l'étape 6, nous gardons dans la classe C uniquement les stations relais qui satisfont les deux contraintes (4.4) et (4.7).
Une fois que la classe de stations relais potentielles C est déterminée, il reste à parcourir chaque bucket de cette classe et de sélectionner la station relais v avec le coût minimum de *trajet-aller-retour*. La méthode de parcours d'un bucket est décrite dans l'algorithme 8.

Algorithme 8 Parcours d'un bucket $Bucket_C(v)$

Entrée: Bucket $Bucket_C(v)$, demande d.
Sortie: *trajet-aller-retour* $c(d, i^\star, j^\star, v)$.

Initialisation : Coût_aller $\leftarrow +\infty$, Coût_aller $\leftarrow +\infty$, $i^\star \leftarrow -1$, $j^\star \leftarrow -1$.
Pour tout i dans $Bucket_C(v)$ **Faire**
  Si o_i et d forment un appariement raisonnable pour le trajet aller avec v comme station relais **Alors**
    Calculer $c(\langle d, i, v\rangle)_{aller}$ comme définie dans (4.9)
    Si $c(\langle d, i, v\rangle)_{aller} <$ Coût_aller **Alors**
      Coût_aller $\leftarrow c(\langle d, i, v\rangle)_{aller}$
      $i^\star \leftarrow i$
    Fin Si
  Fin Si
  Si o_i et d forment un appariement raisonnable pour le trajet retour avec v comme station relais **Alors**
    Calculer $c(\langle d, i, v\rangle)_{retour}$ comme définie dans (4.10)
    Si $c(\langle d, i, v\rangle)_{retour} <$ Coût_retour **Alors**
      Coût_retour $\leftarrow c(\langle d, i, v\rangle)_{retour}$
      $j^\star \leftarrow i$
    Fin Si
  Fin Si
Fin Pour
$c(\langle d, i^\star, j^\star, v\rangle) \leftarrow$ Coût_aller + Coût_retour

Enfin, le meilleur appariement pour une demande d sur le trajet *aller-retour* est déterminé par l'algorithme 9.

Algorithme 9 Ajout d'une demande

Entrée: Graphe $G(V,A)$, demande d.
Sortie: Conducteur $i^\star$, conducteur $j^\star$, meilleur station relais $v^\star$ et $c(\langle d, i^\star, j^\star, v^\star\rangle)$.

Calculer C en utilisant l'**algorithme 7**
Pour tout v **dans** C **Faire**
  Parcourir le bucket $Bucket_C(v)$ en utilisant l'**algorithme 8**
  Garder l'appariement avec le coût minimal $c(\langle d, i^\star, j^\star, v^\star\rangle)$
Fin Pour

Complexité temporelle. Le calcul de complexité repose sur le même principe que celui vu au chapitre précédent. Plus spécifiquement, l'algorithme 9 en utilisant le tas de Fibonacci est de $O(4(|V|\log|V|+|A|)+\sum_{v\in C}|Bucket_C(v)|)$ où $|Bucket_C(v)|$ est le nombre d'offres dans le bucket v.

4.3.3 Suppression d'une offre

La suppression des offres a été décrite dans la section 3.4.2.

4.3.4 Gain total dans le trajet aller-retour

La procédure d'appariement décrite précédemment, assure pour le passager un gain positif simultanément dans le trajet aller et le trajet retour. De cette façon, le partage des économies entre les partenaires du trajet aller est indépendant des partenaires du trajet retour. Mais en réalité, le gain doit être assuré dans la totalité du trajet, pas nécessairement pour le trajet aller et le trajet retour. Par exemple, nous pouvons avoir un gain négatif de -3 € pour le trajet aller entre le passager d et le conducteur i. En revanche, dans le trajet retour, le passager d peut être apparié avec un conducteur j dont les gains s'élèvent à 8 € . Ainsi, le passager peut compenser le gain négatif généré dans son trajet aller. La principale difficulté réside dans la manière de partager les gains (les coûts économisés), qui dépend à la fois des partenaires du trajet aller et également ceux du trajet retour. Il est à noter que la procédure de partage des gains ne sera pas considérée dans cette étude.
Afin de prendre en compte le gain total sur la totalité du trajet aller-retour au lieu du gain séparé, nous devons juste modifier les étapes 3, 10 et 18 dans l'algorithme 8. Plus précisément, dans les deux étapes 3 et 10, nous vérifions seulement la faisabilité des contraintes de *temps de détour* pour le trajet aller et le trajet retour.
Pour finir, dans la ligne 18, on doit assurer la contrainte sur la réduction des coûts entre les trois participants (passager d, conducteur $i^\star$ et conducteur $j^\star$), qui est définie comme suit :

$$c_d(s'\to e')+c_o(s_{i^\star}\to e_{i^\star})+c_d(e'\to s')+c_o(s_{j^\star}\to e_{j^\star})-c(\langle d,i^\star,j^\star,v\rangle)\geq 0 \tag{4.12}$$

Dans le cas où l'utilisateur (conducteur ou passager) souhaite exiger un taux minimum d'économie sur son trajet, l'offre et une demande de covoiturage seront respectivement représentées par o_i=$(s_i,e_i,[t_i^{\min},t_i^{\max}],\Delta_i,\sigma_{o_i})$ et d=$(s',e',[t_{\min}^{aller},t_{\max}^{aller}],[t_{\min}^{retour},t_{\max}^{retour}],\Delta_d^{aller},\Delta_d^{retour},\sigma_d)$, où σ_{o_i} (σ_d) est le pourcentage minimum d'économie de coûts fixés par le conducteur (passager) par rapport à son plus court trajet. Dans ce cas, la contrainte sur les économies de coûts entre le passager d, le conducteur $i^\star$ et le conducteur $j^\star$ sera définie comme suit :

$$\begin{aligned} &c_d(s'\to e')+c_o(s_{i^\star}\to e_{i^\star})+c_d(e'\to s')+c_o(s_{j^\star}\to e_{j^\star})-c(\langle d,i^\star,j^\star,v\rangle)\\ &\geq \sigma_{o_i}\cdot c_o(s_{i^\star}\to e_{i^\star})+\sigma_{o_j}\cdot c_o(s_{j^\star}\to e_{j^\star})+\sigma_d\cdot(c_d(s'\to e')+c_d(e'\to s')) \end{aligned} \tag{4.13}$$

4.4 Attribution de rôles à l'utilisateur

Pour offrir aux utilisateurs d'un système de covoiturage plus de flexibilité, le système pourrait lui-même gérer l'attribution des rôles aux utilisateurs. Plus précisément, au lieu de définir au préalable le rôle de l'utilisateur, le système lui attribue le rôle le plus approprié pour son trajet aller-retour ainsi que la station relais adéquate. Le choix du rôle ainsi que la station relais seront déterminés de manière à ce que les économies en coût soient maximisées. Le rôle de l'utilisateur est validé une fois que les deux rôles (aller et retour) sont attribués. En effet, en attribuant le rôle du passager pour le trajet aller, ce dernier ne peut pas être prévu en tant que conducteur pour son trajet retour et

inversement car les deux cas sont irréalisables en pratique.
L'approche précédente peut être facilement extensible pour rendre le rôle d'un utilisateur flexible. Pour ce faire, deux types de stations relais potentielles seront distinguées :

(i) Des stations relais potentielles pour le rôle du passager.

(ii) Des stations relais potentielles pour le rôle du conducteur.

Pour un conducteur, la station relais pour le trajet aller peut différer de la station relais du trajet retour. Ainsi, les stations relais potentielles du trajet aller et du trajet retour ayant le rôle du conducteur correspondent respectivement aux stations retournées par l'étape 3 (c'est à dire, C_{aller}) et 5 (c'est à dire, C_{retour}) dans l'algorithme 7. Le terme *trajet-aller-retour* $c(\langle r, i, j, v_1, v_2\rangle)$ qui représente le coût du trajet aller-retour pour l'utilisateur r avec le participant i pour le trajet aller (passant par l'emplacement intermédiaire v_1) et le participant j pour le retour (passant par l'emplacement intermédiaire v_2) devient :

$$c(\langle r, i, j, v_1, v_2\rangle) = c(\langle r, i, v_1\rangle)_{aller} + c(\langle r, j, v_2\rangle)_{retour} \tag{4.14}$$

où

$$c(\langle r, i, v_1\rangle)_{aller} = \begin{cases} c_d(s \to v_1) + c_o(s_i \to v_1) & \\ +c_o(v_1 \to e) + c_o(e \to e_i) & \text{if } r = d \quad (4.15a) \\ & \\ c_o(s \to v_1) + c_d(s_i \to v_1) & \\ +c_o(v_1 \to e_i) + c_o(e_i \to e) & \text{if } r = o \quad (4.15b) \end{cases}$$

et

$$c(\langle r, j, e\rangle)_{retour} = \begin{cases} c_o(s_j \to e) + c_o(e \to v) & \\ +c_o(v \to e_j) + c_d(v \to s) & \text{if } r = d \quad (4.16a) \\ & \\ c_o(e \to v_2) + c_d(s_j \to v_2) & \\ +c_o(v_2 \to e_j) + c_o(e_j \to s) & \text{if } r = o \quad (4.16b) \end{cases}$$

Pour résumer, l'objectif ici est plutôt de déterminer le rôle optimal pour l'utilisateur r qui minimise le coût du *trajet-aller-retour* (4.14). Plus précisément, dans le cas où l'utilisateur est affecté en tant que passager (c'est à dire $r = d$), nous devons déterminer une offre o_i pour le trajet aller (4.15a), une station relais v et une offre de o_j pour le trajet retour (4.16a). Pour le trajet retour, le passager sera pris en charge à sa destination ($v_2 = e$). S'il est affecté en tant que conducteur (c'est à dire $r = o$), nous déterminons s'il existe une demande i pour le trajet aller (4.15b) (en passant par la station relais v_1) et une demande j pour le retour (4.16b) (en passant par la station relais v_2). Dans le cas où aucun appariement est trouvé pour le trajet aller-retour, l'utilisateur se déplace seul avec son véhicule pour les trajets aller-retour.
Une question qui peut se poser à ce niveau concerne les trajets allers-retours des participants i et j dans le cas où l'utilisateur courant r est affecté comme conducteur. Plus précisément, l'utilisateur courant r sera apparié avec deux participants i et j si et seulement si leur trajet retour est assuré par le système. Garantir les trajets retours des deux participants i et j rend le système récursif. En effet, si l'utilisateur courant r est affecté comme conducteur et les participants i et j comme passagers, nous devons leur garantir leur trajet retour avec deux nouveaux conducteurs k et m. Et pour les deux nouveaux conducteurs k et m, nous devons également leur garantir deux autres nouveaux passagers et ainsi de suite. Donc, afin d'éviter cette récursivité, nous focalisons uniquement sur le trajet aller-retour de l'utilisateur courant r.

L'attribution du rôle nécessite donc la définition de bucket associé aux passagers passant par une station relais v, noté par $Bucket_P(v)$. Ce dernier permettra de stocker les informations nécessaires des passagers passant par la station relais v et qui n'ont pas pu être appariés lors de leur entrée dans le système. Alors, pour un passager donné d passant par la station relais v, les informations insérées dans le $Bucket_P(v)$ sont :

$$Bucket_P(v) := \{(i, c^{\uparrow}_{s_i}, \tau^{\uparrow}_{s_i}) \mid (\tau^{\uparrow}_{s_i} + \tau^{\downarrow}_{e_i}) \leq (\tau_o(s_i \rightarrow e_i) + \Delta^{aller}_r) \\ \& (\tau^{\uparrow}_{e_i} + \tau^{\downarrow}_{s_i}) \leq (\tau_o(e_i \rightarrow s_i) + \Delta^{retour}_r)\}. \quad (4.17)$$

Les informations du passager d sont insérées dans le bucket $Bucket_P(v)$, si et seulement si le détour encouru par ce dernier en passant par cette station relais pour le trajet aller et retour sont simultanément respectés. En revanche, si le passager souhaite se déplacer uniquement pour un trajet aller, la satisfaction de la contrainte du détour pour le trajet aller suffit.
Ainsi, pour déterminer un meilleur rôle à l'utilisateur r, il ne reste qu'à parcourir ;

- Les *buckets conducteurs* des stations relais appartenant à la classe C afin de sélectionner un conducteur i pour le trajet aller et un conducteur j pour le trajet retour.
- Les *buckets passagers* des stations relais appartenant à la classe C_{aller} pour la détermination d'un passager i pour le trajet aller et la classe C_{retour} pour la détermination d'un passager j pour le trajet retour.

Une fois que le parcours des buckets est achevé, le coût du *trajet-aller-retour* de l'utilisateur r ayant le rôle passager et le rôle conducteur sont déterminés. Nous gardons ainsi le meilleur appariement qui minimise son coût total du *trajet-aller-retour*.

4.5 Expérimentations

Des expériences numériques ont été menées pour une analyse comparative entre les différentes approches proposées. Dans cette section, l'approche du Covoiturage Aller-Retour Classique a été notée par **CARC**. L'approche avec stations relais tout en assurant pour le passager un gain simultané dans le trajet aller et retour a été notée par **CAR**. Enfin, l'approche avec stations relais tout en assurant pour le passager un gain global en aller-retour a été notée par **CARG**. Les résultats ont été évalués en termes de coûts économisés, de nombre d'appariement et de temps de calcul.

Environnement. Les algorithmes ont été codés en C# avec une machine équipée d'un processeur Intel (R) Core (TM) I7-3520M CPU 2,90 GHz et 8 Go de mémoire RAM. Un tas binaire a été utilisé comme la structure de données de la file de priorité.

Données réelles. Nous avons utilisé les mêmes données réelles décrites dans la section 3.5.2 en considérant les trajets aller-retour, c'est-à-dire domicile-travail et travail-domicile. Pour chaque offre aller-retour, nous produisons deux offres séparées, une pour le trajet aller (de son domicile à son lieu de travail) et une autre pour le trajet retour (de son lieu de travail à son domicile). Ainsi, le nombre d'offres générées s'élève à 1512. En ce qui concerne les demandes, chaque passager possède une voiture personnelle qu'il peut utiliser pendant une partie de son voyage. L'ensemble des offres et des demandes est filtré de manière que nous ne pouvons pas trouver une offre et une demande qui ont à la fois le même lieu de départ et la même destination. La fenêtre de temps et le temps de détour de chaque trajet fixé par chaque employé sont :

- Pour le trajet aller, l'heure de départ au plus tôt et au plus tard sont respectivement fixées à $07h30$ et à $08h00$.

– Pour le trajet retour, l'heure de retour au plus tôt et au plus tard sont respectivement fixées à 18h00 et 18h15.
– Le temps de détour du conducteur (passager) est fixé au plus à 20% de la durée de son trajet direct.

Réseau routier. Le même réseau routier décrit dans 3.5.2 est utilisé. Chaque nœud de ce réseau est considéré comme une station relais potentielle dans laquelle le conducteur et le passager peuvent se rencontrer.

Résultats numériques. Pour faciliter notre analyse, nous avons partitionné l'ensemble des offres et des demandes en six classes principales dont chaque classe représente une ville géographique (voir figure 4.3).

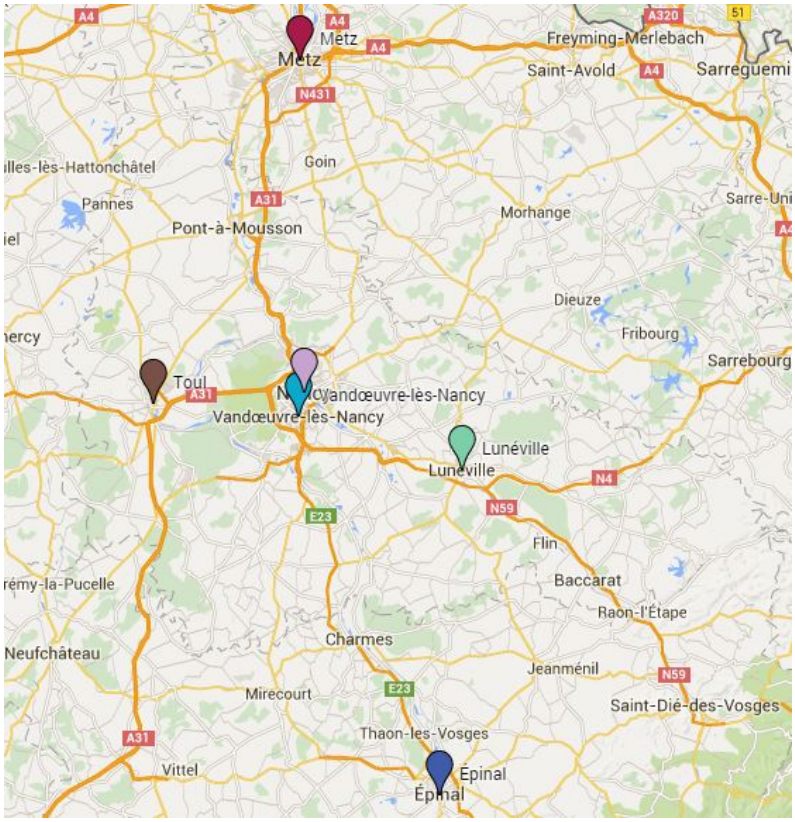


FIGURE 4.3 – Les six classes principales (Nancy, Vandoeuvre-lès-Nancy, Metz, Lunéville et Épinal).

Pour chaque classe, nous avons mesuré les paramètres suivants qui nous ont permis de comparer l'efficacité des solutions des différentes approches.

– Taux de réussite moyen des appariements ($Match$) : le nombre de demandes appariés divisé par le nombre total de demandes au sein de chaque classe, c'est à dire, $\sum_{i=0}^{n_j} \frac{m_i}{n_j} \cdot 100$ ($m_i = 1$ si l'appariement est détecté par l'heuristique ou le covoiturage classique, 0 sinon, et n_j est le nombre total de demandes dans la classe j).
– Taux moyen d'économie en aller-retour (CE) : la somme des taux d'économie des coûts en aller-retour générée par les appariements dans la classe divisée par le nombre total de demandes appariées dans chaque classe, c'est-à-dire, $\sum_{d \in classe_j} \frac{cs_i + cs_j - (c_o(s \to e) + c_o(e \to s))}{c_o(s \to e) + c_o(e \to s)}$ avec cs_i et cs_j le

coût du trajet associé à la demande d respectivement pour le trajet aller (avec le conducteur i) et retour (avec le conducteur j).

Classes (Villes)	# demandes	**CARC**	**CAR**		**CARG**	
		$Match$ (%)	$Match$ (%)	CE (%)	$Match$ (%)	CE (%)
Nancy	311	94.2	97.1	35.2	97.1	35.2
Vandoeuvre-lès-Nancy	199	96.4	98.4	32.7	98.4	32.7
Metz	38	68.4	94.7	22.7	97.3	21.8
Lunéville	107	95.3	99	25.7	99	25.7
Toul	74	91.8	97.2	27.8	97.2	27.8
Épinal	28	60.7	82.1	21.2	89.2	20.2

TABLE 4.1 – Performances des trois approches.

Les résultats du tableau 4.1 montrent l'efficacité de **CAR** et **CARG** par rapport à **CARC** en termes d'appariement réussis ($Match$) sur les six classes.

L'écart du succès en termes d'appariement entre les **CARC** et **CAR** (**CARG**) diminue avec la densité des offres dans les classes concernées. Par exemple, le gap entre **CARC** et **CAR** (**CARG**) atteint 26,3% (28,9%) pour la ville de Metz et 21,4% (28,5%) pour la ville d'Épinal. En effet, lorsque la densité des offres est faible ou éloignée des demandes, cela oblige les conducteurs à faire de longs détours pour la prise en charge des passagers. Malheureusement, ces derniers sont limités par le temps et le coût de détour d'où l'échec de quelques appariements. Néanmoins, lorsque le passager accepte de se déplacer avec sa propre voiture à une station relais (approches **CAR** et **CARG**), cela permet de réduire le détour des conducteurs, d'où un nombre d'appariement réussis plus élevé. Il est à noter que dans **CAR** et **CARG**, le taux d'appariement réussis est le même dans toutes les classes, sauf dans les classes de Metz et Épinal où **CARG** surpasse **CAR** avec un léger ratio qui n'excède pas 7.1%.

Concernant la ville d'Épinal, les économies de coûts (CE) engendrés par **CARG** sont moins élevées que celles dans **CAR**. Ceci est dû au fait que les économies de coûts moyens (CE) sont calculées sur l'ensemble des appariements réussis alors que **CARG** détecte plus d'appariement que **CAR**.

Pour avoir une meilleure idée sur les performances de **CARC**, **CAR** et **CARG** en termes d'économies de coût, nous évaluons les économies des coûts supplémentaires que nos approches génèrent par rapport au covoiturage classique **CARC** sur les instances (notées par I). Ces instances regroupent les appariements qui sont réussis par les trois approches **CARC**, **CAR** et **CARG**. Nous comparons également les deux approches **CAR** et **CARG** sur des instances (notées par H) où un appariement est réussi seulement par ces deux dernières.

Les résultats sont présentés dans le tableau 4.2. Dans la colonne C_I et C_H, les économies moyennes des trajets aller-retour sur les instances I et H sont présentées.

Classes	C_I			C_H	
	CARC (%)	**CAR** (%)	**CARG** (%)	**CAR** (%)	**CARG** (%)
Nancy	31	36.2	36.2	35.2	35.2
Vandoeuvre-lès-Nancy	29.7	33.3	33.3	32.7	32.7
Metz	25	32.3	32.3	22.7	22.7
Lunéville	25.9	26.8	26.8	25.7	25.7
Toul	26.4	29.4	29.4	27.8	27.8
Épinal	25	28.7	28.7	21.2	21.2

TABLE 4.2 – Économies supplémentaire générées.

D'après le tableau 4.2, nous pouvons constater que les économies en coûts générées sur toutes les classes sont plus importantes dans les deux approches **CAR** et **CARG** par rapport au **CARC**. En effet, le rapport coût-épargne moyen par trajet aller-retour atteint 36,2% en considérant une station relais comme lieu de prise en charge.
En outre, les deux approches **CAR** et **CARG** ont les mêmes économies pour les appariements détectés. Dans le cas où un appariement est uniquement détecté par **CARG**, les économies sur le trajet aller (resp. retour) ne pourront pas excéder 25% du trajet aller-retour. En effet, pour un trajet aller (resp. retour), nous ne pouvons pas économiser plus de 25% du trajet aller-retour. Or, comme l'approche **CAR** ne détecte pas d'appariement, ceci implique un gain négatif sur un des trajets (trajet aller ou trajet retour). Ainsi, les économies que **CARG** génèrent dans le cas où **CAR** ne détecte pas d'appariement, ne dépassent pas 25%.
Finalement, le principal avantage de l'approche **CARG** par rapport à **CAR** réside dans le nombre d'appariement réussis tout en conservant une économie attractive.

Pour illustrer l'efficacité de l'ensemble du système, nous évaluons le temps d'ajout d'une demande en fonction du nombre d'offres et de buckets dans le système.
Le tableau 4.3 montre le temps nécessaire pour ajouter une demande en fonction du nombre d'offres et de buckets. Chaque entrée dans le tableau 4.3 représente une valeur moyenne sur l'ensemble des instances de chaque classe.

Classes	# de buckets		# offres dans les buckets $[\cdot 10^3]$			Temps de calcul [s]		
	CARC	**CAR**	**CARC**	**CAR**	**CARG**	**CARC**	**CAR**	**CARG**
Nancy	2	285	0.064	48	48	0.19	0.44	0.44
Vandoeuvre-lès-Nancy	2	374	0.090	47	47	0.24	0.55	0.56
Metz	2	2442	0.010	39	39	2.77	3.39	3.52
Lunéville	2	1700	0.043	44	44	0.55	1.73	2.03
Toul	2	832	0.041	48	48	0.49	1.45	1.51
Épinal	2	3740	0.006	31	31	3.5	5.29	5.45

TABLE 4.3 – Ajout d'une demande.

D'après le tableau 4.3, le temps moyen d'ajout d'une demande croit linéairement avec le nombre de buckets ainsi que le nombre d'offres dans les buckets. Pour l'approche **CARC**, nous devons seulement parcourir les deux buckets correspondant aux lieux de départ et de destination de la demande.

Cependant, pour **CAR** et **CARG**, nous devons analyser tous les buckets se trouvant dans la classe *de stations relais potentielles C*. La légère différence en temps de calcul entre **CAR** et **CARG** (< 0.2 sec) est dû à la façon dont on parcourt un bucket. Contrairement à l'approche **CAR**, dans **CARG** nous vérifions la *contrainte d'économie* qu'après avoir parcouru toutes les offres contenues dans le bucket qui satisfait la contrainte du *temps du détour*. Ainsi, le nombre d'offres filtrés dans le bucket est moins important par rapport à l'approche **CAR**. Pour résumer, les deux approches de la **CAR** et **CARG** fournissent des solutions efficaces en termes d'appariement réussis, taux d'économie de coûts et le temps de calcul. Elles résolvent efficacement le problème covoiturage aller-retour avec un nombre élevé de stations relais tout en ayant un délai de réponse raisonnable.

4.6 Conclusion du chapitre

Dans ce chapitre, nous avons proposé des méthodes pour le problème de covoiturage aller-retour qui permettent aux passagers d'utiliser leur voiture privée afin d'atteindre une station relais plus ou moins proche de leur emplacement de départ. Le processus de prise de décision sur le choix de la station relais se rapporte à la densité des offres potentielles passant par cet emplacement. Les approches ont été validées par des expérimentations fondées sur un jeu de données réelles de covoiturage avec un vrai réseau routier de la région Lorraine. Les avantages principaux de ces approches sont l'augmentation du nombre de succès d'appariement entre conducteurs-passagers pour les trajets de type aller-retour, auquel s'ajoute une réduction significative du coût du trajet total par rapport à l'approche classique de covoiturage aller-retour.

5

Dynamic Dial-a-Ride appliqué aux taxis-partagés et au covoiturage dans un milieu urbain

Sommaire

Dans ce chapitre, nous présentons des heuristiques d'insertion pour un problème dynamique de type Dial-a-Ride with Time Window, conçu pour une application de partage de taxis en temps réel. L'application s'appuie sur des cartes géographiques avec les coordonnées GPS en temps réel des taxis et des clients. Un client envoie une demande de trajet à une centrale de taxis, qui lui attribue en temps quasi-réel le taxi le plus approprié. Elle tient compte de la capacité du taxi, de la contrainte de coût ainsi que des contraintes temporelles : le temps d'attente maximum avant la prise en charge du client et la durée maximale du trajet pour chaque passager. Les passagers peuvent partager leur taxi pour une partie ou la totalité de leur voyage.
Nous présentons ensuite quelques extensions de notre modèle, à savoir : la prise en compte des lieux intermédiaires de prise en charge et de dépose des clients. Dans un second temps, nous développerons une approche qui considère une flotte mixte de taxis et de covoiturage. La méthodologie repose sur le calcul des plus courts chemins en temps-réel, sur la détermination des bornes inférieures valides, suivi d'un algorithme d'insertion avec contrôle de validité. Chaque demande d'un nouveau client est donc insérée dans un itinéraire. Des expériences sur des données réelles de taxis New-Yorkais ainsi qu'un vrai réseau routier montrent la viabilité de nos méthodes.

5.1 Introduction

Dans ce chapitre, nous nous concentrons sur une compagnie de taxis qui veut optimiser ses coûts quotidiens de fonctionnement en mettant en œuvre un nouveau modèle d'affaires permettant aux clients de partager un même taxi. L'idée est la suivante : alors qu'un client est amené à sa destination dans un taxi, une demande d'un nouveau client arrive. Il peut alors être plus efficace d'ajuster la trajectoire du taxi afin d'y inclure le nouveau client sans nécessairement solliciter un nouveau taxi vide. Chaque client spécifie son lieu de prise en charge et sa destination. Le taxi le plus approprié doit lui être attribué afin de répondre aux demandes en supposant que plusieurs clients peuvent partager simultanément un même taxi sur certaines parties de leur voyage. En outre, afin de tenir compte de la satisfaction de chaque client, des contraintes spécifiques doivent être remplies comme : le temps d'attente maximum avant la prise en charge, la durée maximale du trajet et son coût. Comme les demandes des clients arrivent aléatoirement pendant la journée et doivent être satisfaites en temps réel, notre problème est un problème de type Dynamic Dial-A-Ride with Time Window (DDARPTW). Dans la littérature ce type de problème n'est pas associé à la difficulté de trouver des plus courts chemins, pourtant nécessaires à l'inclusion de nouveaux clients [Cordeau and Laporte, 2007, Berbeglia et al., 2010]. Au lieu de cela, il est supposé que tous les plus courts chemins entre tous les emplacements possibles peuvent être pré-calculés et une matrice de distance constante peut être utilisée pour résoudre algorithmiquement le DARP dynamique avec fenêtre de temps. Cependant, une telle approche a des limites pratiques : les réseaux routiers réels sont dynamiques et le temps de parcours pour tout segment de route peut changer de façon significative au cours de la journée (c'est à dire entre les heures de pointe et les heures creuses), les demandes peuvent survenir de n'importe quelle position (pas forcément de l'un des sommets pré-traité). Comme de telles circonstances peuvent entraîner des pertes significatives de la qualité d'une solution, nous trouvons insuffisant d'utiliser l'habituelle distance (resp. durée) constante de l'approche matricielle. C'est pourquoi nous développons une heuristique basée sur l'algorithme de Dijkstra, A* et la recherche bidirectionnelle, permettant les calculs des plus courts chemins, ainsi que sur un algorithme d'insertion [Coslovich et al., 2006, Häme, 2011]. En utilisant les contraintes du DDARPTW afin de réduire l'espace de solutions, nos heuristiques sont capables de trouver des solutions en temps (quasi) réel pour des problèmes dont la taille est celle d'une ville, comprenant le calcul des plus courts che-

mins. Les articles traitant le problème DARP ont été résumés dans [Cordeau and Laporte, 2007] et [Berbeglia et al., 2010]. Dans cette étude, nous nous intéressons au problème suivant : étant donné une nouvelle demande du trajet d'origine s à une destination e, quels taxis ou covoiturage peuvent satisfaire cette demande de sorte qu'un temps d'attente maximum avant la prise en charge, la durée maximale de trajet ainsi que le coût maximal toléré soient respectés pour tous les passagers ?
La première approche proposée prend en compte uniquement les taxis (compagnie de taxis). La deuxième approche se positionne dans un cadre plus général où l'entreprise est composée de taxis et d'offres de covoiturage. Les deux approches sont basées sur une détermination d'un ensemble de plus courts chemins. Dans certaines situations, le coût à payer par un passager sur un voyage partagé peut être plus cher que son coût s'il voyageait seul avec le conducteur. Nous pouvons considérer ce critère comme une contrainte qui assure aux passagers un coût de trajet plus attractif en partageant un véhicule plutôt que de voyager séparément. Le modèle présenté dans [Santos and Xavier, 2015] prend en considération le coût de voyage des passagers. Plus spécifiquement, un passager est affecté à un taxi seulement si son nouveau coût est inférieur ou égal à son coût direct s'il voyageait seul. Une fois que le passager est affecté au taxi, le coût de référence associé à ce passager reste inchangé (ce coût de référence correspondra à son coût initial s'il voyageait seul). Ceci implique que les coûts des passagers à bord peuvent être augmentés par rapport aux coûts annoncés avant d'autres insertions éventuelles. Dans notre approche, le coût de référence associé aux passagers correspond à leur nouveau coût à chaque nouvelle insertion. Cela implique que l'ajout d'un nouveau client est possible uniquement si les coûts des passagers à bord n'augmentent pas par rapport à leur nouveau coût de référence. Le reste du chapitre est organisé comme suit : la section 5.2 décrit le problème de prédiction de l'emplacement d'un véhicule en mouvement dont la solution permet d'estimer avec précision la position des véhicules. La section 5.3 nous donne un aperçu des applications pratiques de notre modèle en présentant la fonction objectif choisie pour notre application. Dans la section 5.4, nous décrivons le processus de prise en compte du coût du trajet tout en mettant à jour le coût de référence des passagers à chaque nouvelle insertion. La section 5.5 décrit la première méthode que nous utilisons pour résoudre le DDARPTW dans le cas où la flotte est composée uniquement de taxis. Dans la section 5.6, nous proposons une deuxième méthode qui permet de résoudre le DDARPTW dans le cas où la flotte est composée de taxis ainsi que des offres de covoiturage. Dans la section 5.7, nous expliquons les spécificités de chacune des deux approches. Des expériences sur des données réelles et sur un vrai réseau routier sont présentées dans la section 5.8. Enfin, la section 5.9 résume nos réalisations et présente les limites de notre modèle.

5.2 Hypothèses sur le positionnement des véhicules

Les GPS embarqués dans les taxis envoient leurs coordonnées géographiques, généralement à des intervalles de temps réguliers. Cependant, des problèmes techniques telle que la perte du signal lors du passage dans un tunnel, l'expiration de la batterie, l'application qui s'éteint, peuvent empêcher l'envoi de cette information. L'emplacement précis de la position actuelle du taxi doit donc être estimée. Formellement, le problème de prédiction des trajectoires sur un réseau routier peut être défini sur la base de Eisner [Eisner et al., 2011] : Soit $G = (V, A, W)$ un réseau routier sur un ensemble V de nœuds, A un ensemble d'arcs, et une fonction w sur l'ensemble des arcs qui donne une mesure associée à un arc, généralement une durée ou une longueur. Compte tenu d'un chemin $P = v_{t_0}, v_{t_1}, \ldots, v_{t_i}$ qui donne les localisations aux temps $t_0 < \ldots < t_i$ d'un taxi, quelles sont les localisations de ce taxi aux temps $t_{i+1} < \ldots < t_{i+k}$? Basé sur des données historiques, nous pourrions estimer la probabilité d'atteindre certaines localisations en fonction des dernières positions GPS envoyées, comme cela est fait dans [Liu et al., 2010, Kim et al., 2007]. Dans notre application, nous n'avons pas de

telles données historiques sur le long terme, mais seulement sur la journée. Nous utilisons le schéma suivant pour résoudre ce problème : soit un conducteur n'a pas défini de destination, ce qui signifie qu'il n'a pas de client, soit le taxi a déjà une destination. Dans le premier cas, il n'y a généralement pas d'autres informations que sa précédente position GPS. La seule estimation fiable de la position consiste à observer son environnement. Par exemple, si les dernières positions du conducteur sont sur une autoroute, ou le long d'une rue à sens unique, la meilleure estimation de la position possible est vers la jonction suivante ou la sortie d'autoroute suivante. Si la différence entre l'heure actuelle et la dernière position GPS est faible, connaitre la limitation de vitesse de la route et en formulant une hypothèse sur la vitesse du taxi, il est possible d'estimer sa position actuelle. Dans le deuxième cas, lorsque l'itinéraire du taxi et ses arrêts sont connus, l'hypothèse suivante est faite : le taxi suit un ensemble de plus courts chemins entre chacun de ses arrêts. Il est donc possible d'estimer sa position, avec une hypothèse sur la vitesse du taxi sur différents types de routes. Le procédé consiste à exécuter un algorithme de plus court chemin pour trouver les couples (position, temps) suivant son itinéraire le plus rapide.

5.3 Fonction objectif

Dans ce qui suit, nous notons un plus court chemin de a vers b comme $a \rightsquigarrow b$, et sa durée par $\delta(a,b)$. Chaque requête d'un client est caractérisée par un lieu de départ s, une destination e, un temps d'attente maximal avant la prise en charge $maxwait$ et un instant d'arrivée maximal à sa destination $t_{\max}$.
La définition de la fonction objectif dépend du but recherché par le gestionnaire. La qualité de service peut être améliorée selon plusieurs critères :

- Le temps d'attente avant la prise en charge peut être minimisé.
- Le temps moyen d'arrivée à destination peut être minimisé. Cet objectif peut aussi être adapté, par exemple en prenant la somme des différences au carré, afin qu'aucun passager ne reste trop longtemps dans le taxi.
- Le temps d'arrivée à destination du dernier passager peut être minimisé, de sorte que la durée d'itinéraire du taxi soit la plus courte possible.

Dans ce chapitre, la fonction objectif utilisée consiste à minimiser le temps de détour relatif effectué par le conducteur pour transporter le nouveau client. Autrement dit, le nouveau client est inséré dans une route d'un taxi (notée P) qui minimise son *coût d'insertion relatif* associé à cette route, noté par la fonction $f_r(P,i,j)$. Plus précisément, le lieu de prise en charge du nouveau client est inséré entre l'arrêt i et l'arrêt $i+1$, et sa position de dépose entre l'arrêt j et l'arrêt $j+1$.

$$f_r(P,i,j) = \frac{\delta(i,s)+\delta(s,i+1)-\delta(i,i+1)+\delta(j,e)+\delta(e,j+1)-\delta(j,j+1)}{\delta(P)}.$$

5.4 Prise en compte du coût du trajet

Comme mentionné ci-dessus, nous prenons en compte les coûts des trajets des passagers. Plus spécifiquement, nous garantissons aux passagers l'attractivité en termes de coût de partage pour un même taxi. Ainsi, pour chaque insertion d'un nouveau client dans un taxi, nous devons vérifier l'attractivité en coût pour les différents passagers à bord et en cours de prise en charge. Autrement dit, nous devons vérifier si l'ajout de ce nouveau client dans le taxi n'augmente pas le coût du trajet des différents passagers. Dans ce qui suit, l'indice '+' sur un noeud X correspond au lieu de prise en charge du passager X, tandis que le lieu de dépose est représenté par l'indice '—'.
Ainsi, on retrouvera quatre types de passager :

(i) Les passagers pris en charge avant le nouveau client et qui sont déposés avant la dépose du nouveau client, notés $< +, s, -, e >$.
(ii) Les passagers pris en charge avant le nouveau client et qui sont déposés après la dépose du nouveau client, notés $< +, s, e, - >$.
(iii) Les passagers pris en charge après le nouveau client et qui sont déposés avant la dépose du nouveau client, notés $< s, +, -, e >$.
(iv) Les passagers pris en charge après le nouveau client et qui sont déposés après la dépose du nouveau client, notés $< s, +, e, - >$.

Dans la figure 5.1 nous présentons la trajectoire d'un taxi avant et après l'insertion d'un nouveau client r.

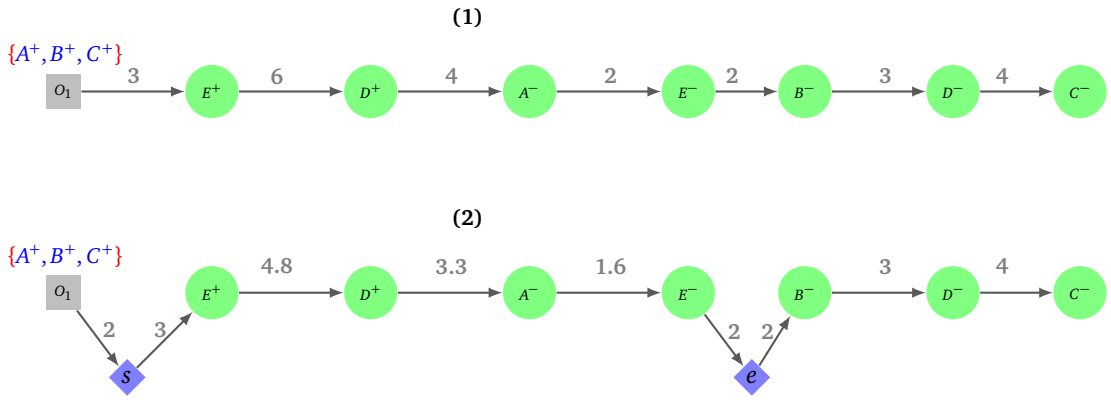


FIGURE 5.1 – La sous-figure (1) représente la trajectoire d'un taxi avant l'insertion du nouveau client r. À l'instant t_0 où le nouveau client r entre dans le système, le taxi se trouve à la position actuelle O_1 ayant déja à bord trois passagers : A, B et C. Les autres arrêts constituant son trajet correspondent aux futurs lieux de prise en charge (E^+, D^+) et de dépose $(A^-, E^-, B^-, D^-, C^-)$ des passagers qui sont déja planifiés avant l'arrivée du nouveau client r (c'est-à-dire, avant l'instant t_0). En revanche, la nouvelle trajectoire du taxi après l'insertion du nouveau client est représentée par la sous-figure (2). Le nouveau lieu de prise en charge s est inséré entre O_1 et E^+, tandis que le nouveau lieu de dépose e est inséré entre E^- et B^-.

Dans cette dernière, nous illustrons les quatre types de passagers évoqués ci-dessus.
L'attractivité des nouveaux coûts attribués aux passagers $< s, +, -, e >$ après l'insertion du nouveau client n'est pas nécessaire. En effet, leur trajet reste inchangé car leur prise en charge s'effectue après celle du nouveau client et leur dépose s'effectue avant la dépose du nouveau client. En revanche, leur nouveau coût ne peut que décroître car le nouveau client participera aux frais de ces trajets inchangés. Concernant les passagers $< +, s, -, e >$, $< s, +, e, - >$ et $< +, s, e, - >$, leur nouveau coût après l'insertion du nouveau client n'est pas forcément attractif comparé à leur ancien coût. Pour les passagers $< s, +, e, - >$ (exemple du passager D dans la figure 5.1), l'attractivité des nouveaux coûts attribués sera assurée par une seule et unique contrainte. Pour les passagers $< +, s, -, e >$ et $< +, s, e, - >$ (exemple des passagers A et C), ces attractivités devront être vérifiées une par une.
La figure 5.2 illustre les vérifications effectuées sur les nouveaux coûts des différents types de passager. Le coût associé à chaque arc correspond au prix par passager et qui n'est autre que le prix du trajet divisé par le nombre de passagers entre les deux arrêts. En effet, le coût d'un segment de trajet est divisé équitablement entre les différents passagers. Comme on peut le constater, la vérification de l'attractivité du nouveau coût associé au passager E est assurée sans effectuer aucune vérification car

son trajet depuis son lieu de départ E^+ jusqu'à sa destination E^- n'a pas changé. Il est à noter que les deux termes "*Ancien coût*" et "*Nouveau coût*" dans la figure 5.2 représentent respectivement l'ancien et le nouveau coût d'un passager sur une partie de son trajet (pas nécessairement sur la totalité de son trajet). En effet, l'attractivité du nouveau coût peut être assurée directement sur cette partie du trajet. Par exemple pour le passager D, son nouveau coût est attractif si et seulement si le nouveau coût d'une partie de son trajet (c'est à dire depuis D^+ jusqu'à B^-, est moins cher que son ancien coût sur la partie D^+ jusqu'à B^-).

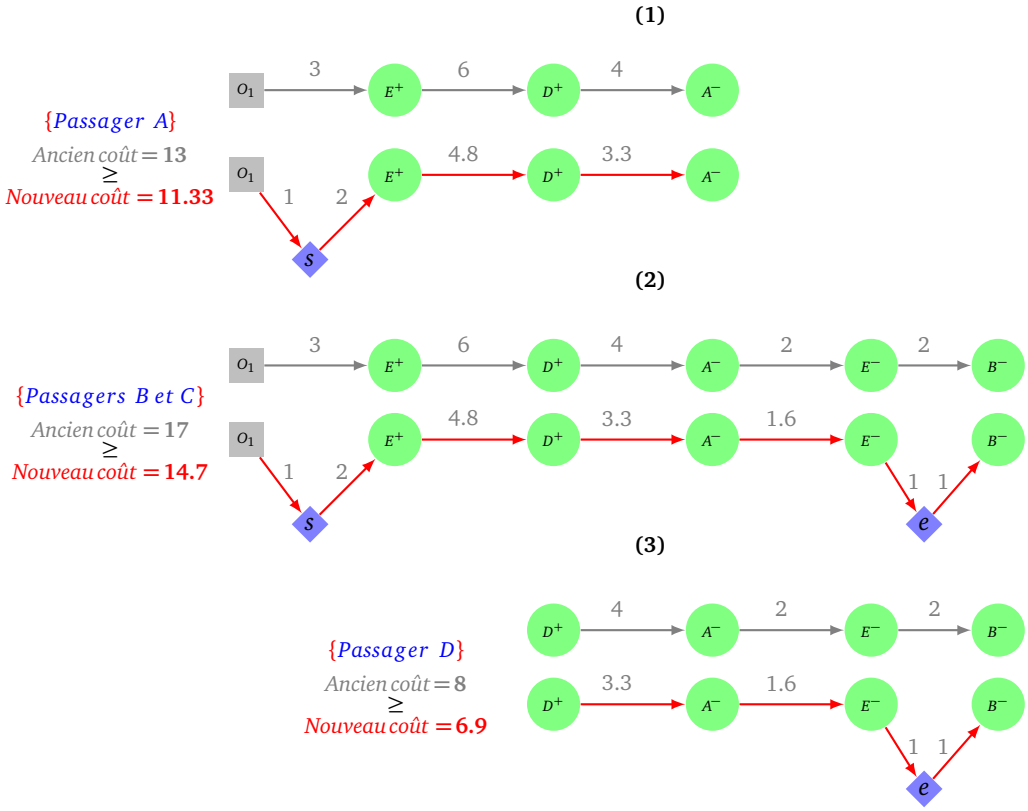


FIGURE 5.2 – Les différentes vérifications effectuées des nouveaux coûts associés respectivement aux passagers $< +, s, -, e >$, $< +, s, e, - >$ et $< s, +, e, - >$ sont réprésentées par trois sous-figures. La sous-figure (1) assure l'attractivité sur les nouveaux coûts attribués aux passagers $< +, s, -, e >$, autrement dit le passager A. La sous-figure (2) assure l'attractivité des coûts attribués aux passagers $< +, s, e, - >$, qui correspondent dans ce cas aux passagers B et C. En revanche, la sous-figure (3) assure l'attractivité des coûts attribués aux passagers $< s, +, e, - >$, ce qui correspond au passager D.

5.5 Première approche considérant uniquement les taxis

Dans cette première approche, la flotte est composée uniquement de taxis. La première étape consiste à trouver un ensemble de plus courts chemins vers le nouveau lieu de prise en charge s. Autrement dit, tous les taxis en mesure de prendre en charge le nouveau client de son lieu de départ s dans un délai d'attente maximum et en respectant la capacité du véhicule, sont découverts. Ensuite, trois autres ensembles de plus courts chemins sont calculés. Ces derniers pourront être utilisés comme

composants de nouveaux itinéraires pour les taxis précédemment découverts. Enfin, les nouveaux itinéraires mentionnés précédemment sont construits par une procédure d'insertion en vérifiant leur admissibilité. Les quatre ensembles de plus courts chemins sont décrits ainsi :

1. Prise en charge des taxis vers l'origine de la demande.
2. Origine de la demande vers sa destination et les arrêts des taxis : de l'origine de la demande vers sa destination et les arrêts d'un taxi.
3. Destination vers les arrêts des taxis : de la nouvelle destination vers les arrêts de prise en charge et de dépose des passagers d'un taxi.
4. Arrêts des taxis vers la destination : des arrêts de prise en charge et de dépose des passagers d'un taxi vers la nouvelle destination.

Les plus courts chemins résultant de ces quatre étapes sont respectivement $P_{\cdot \rightsquigarrow s}, P_{s \rightsquigarrow \cdot}, P_{\cdot \rightsquigarrow e}$ et $P_{e \rightsquigarrow \cdot}$. Ces plus courts chemins sont ensuite combinés pour former un nouvel itinéraire pour un taxi tout en vérifiant l'admissibilité en termes de : temps d'attente maximum avant la prise en charge, la capacité du taxi, la durée maximale et le coût du trajet pour chaque passager.

5.5.1 Prise en charge $P_{\cdot \rightsquigarrow s}$

Le premier problème à résoudre consiste à sélectionner des taxis capables de répondre à une demande dans un délai d'attente maximum $maxwait$ avant la prise en charge tout en respectant la capacité du véhicule. Une façon d'obtenir une telle sélection de taxis pourrait être d'appliquer nb fois un algorithme du plus court chemin, nb étant le nombre de taxis présents au lieu de prise en charge du client. Cela signifie qu'un algorithme de plus court chemin doit être appliqué à chaque nouvelle demande et depuis chaque position d'un taxi. Sachant que la fréquence de nouvelles demandes dans les grandes villes peut facilement atteindre plusieurs dizaines par seconde, une telle méthode n'est pas efficace. Dans notre approche, la sélection de taxis est basée sur un algorithme de plus court chemin de type un-vers-plusieurs, calculée sur le graphe inversé depuis le lieu de prise en charge vers toutes les positions des taxis. Notre algorithme du plus court chemin est basé sur l'algorithme de Dijkstra pour plusieurs raisons. D'abord, le problème est limité par la taille de l'agglomération puisque nous considérons une compagnie de taxis opérant dans une ville. Deuxièmement, nous utilisons un véritable réseau routier au lieu d'une forme compacte comme c'est le cas pour les algorithmes les plus performants de plus courts chemins, tels que ceux décrits par [Sanders and Schultes, 2005, Geisberger et al., 2012]. Cela nous permet d'utiliser des informations en temps réel telles que les restrictions routières temporaires, les embouteillages, les limitations de vitesse temporelles, etc. L'adaptation de l'algorithme de Dijkstra est effectuée comme suit :

– Une limite $maxwait$ représente le temps d'attente maximum autorisé avant la prise en charge. Cette limite est utilisée comme critère d'arrêt : si le nœud actuellement pris en compte dans l'algorithme de Dijkstra a une étiquette de durée supérieure à $maxwait$, alors il n'y a pas de chemin à partir de ce noeud jusqu'au lieu de prise en charge dont l'étiquette de durée est inférieure ou égale à $maxwait$.

Durant la procédure de recherche, trois types de nœuds sont susceptibles d'être marqués :

– Un simple nœud v correspondant à un nœud du réseau.
– Un nœud v correspondant à une position actuelle d'un taxi z. Si la capacité maximale du véhicule n'est pas excédée, alors le taxi est inséré dans la liste des taxis candidats car il peut atteindre la position de prise en charge à l'intérieur des $maxwait$.
– Un nœud v correspondant à un des arrêts d'un taxi z. Dans ce cas, avant d'insérer ce taxi dans la liste des candidats, nous devons non seulement vérifier la faisabilité en termes de capacité du taxi, mais également la faisabilité en temps d'arrivée au lieu de prise en charge en passant par v.

Autrement dit, nous devons vérifier la faisabilité de cette contrainte $t_v^z + \delta(v,s) \leq t_0 + maxwait$ (avec t_0 le moment où la demande à été envoyé par le client c_r et t_v^z est le temps d'arrivée du taxi z à l'arrêt v).

Dans le pseudo-algorithme 10, nous détaillons les opérations effectuées dans l'étape de prise en charge.

Algorithme 10 DijkstraInverseBorné$(G, s, maxwait)$

Entrée: Graphe $G = (V,A)$, source s, Durée $maxwait$
Sortie: Les plus courts chemins à partir des positions/arrêts des taxis potentiels vers le lieu de prise en charge s.
$Liste_{AP}$: Liste des arrêts de taxis potentiels pour la prise en charge.
$Liste_{RP}$: Liste des routes des taxis potentiels.

Initialisation : $Q.Inserer(s)$, $Liste_{AP} \leftarrow \emptyset$, $Liste_{RP} \leftarrow \emptyset$, *Recherche* $\leftarrow$ Vrai
{Q est la file de priorité}
Tant que *Recherche* $=$ Vrai **Faire**
  $v \leftarrow Q.\min()$
  Si $\delta(s,v) > maxwait$ **Alors**
    Recherche $\leftarrow$ Faux
    STOP
  Fin Si
  Marquer le noeud v comme visité et retirer ce dernier de la file de priorité Q.
  Si v est une position actuelle d'un taxi z **ou** $\big(v$ est un arrêt d'un taxi z **et** $t_v^z + \delta(v,s) \leq t_0 + maxwait\big)$ **Alors**
    Si la capacité maximale du taxi à l'arrêt v n'est pas atteinte **Alors**
      $Liste_{AP} \leftarrow Liste_{AP} \cup \{v\}$
      {Soit P l'itinéraire du taxi z}
      $Liste_{RP} \leftarrow Liste_{RP} \cup \{P\}$
    Fin Si
  Fin Si
  Mettre à jour les coûts provisoires des voisins du noeud v et *insérer* ces derniers dans la file de priorité Q.
Fin Tant que

5.5.2 Origine de la demande vers sa destination et les arrêts des taxis $P_{s\rightsquigarrow.}$

Un taxi capable de prendre en charge un nouveau client c_r peut ne pas avoir le temps de l'emmener directement à sa destination et ensuite servir ses autres clients $c_1, c_2, \ldots, c_{r-1}$. Dans ce cas, il pourrait être plus efficace après la prise en charge du nouveau client, de conduire d'abord vers les destinations/origines de ses clients $c_1, c_2, \ldots, c_{r-1}$ avant de rejoindre la destination e du nouveau client c_r. Nous utilisons une version modifiée de l'algorithme A* du lieu de prise en charge s de la nouvelle demande r vers sa destination e. L'algorithme A* s'arrête lorsque la destination est atteinte. Par conséquent, nous le modifions afin de pouvoir effectuer des détours. Plus exactement, nous définissons λ_r le coefficient de majoration sur la durée d'un trajet le plus rapide $\delta(s,e)$ entre s et e. Une fois que l'algorithme original A* a atteint la destination e, $\delta(s,e)$ est connu et l'algorithme continue jusqu'à ce que tous les nœuds atteints aient une durée d'étiquette supérieure ou égale à $\lambda_r\delta(s,e)$. Nous avons fixé $\lambda_r = 1.2$, ce qui signifie que tous les clients acceptent une durée du voyage allongée d'au plus 20% par rapport à l'itinéraire le plus rapide. De cet algorithme, la plus grande durée jusqu'à la destination D_r^{dest} du c_r est calculée comme suit

$$D_r^{dest} = \lambda_r \delta(s,e)$$

Tandis que l'heure d'arrivée la plus tardive est calculée de cette façon

$$t_{\max r} = t_0 + maxwait + D_r^{dest}$$

avec t_0 le moment où la demande a été envoyée par le client c_r. Notons qu'une fois que le client r est affecté à un taxi z, nous mettons à jour son heure d'arrivée la plus tardive $t_{\max r}$ à $t_s^z + D_r^{dest}$. Les routes des taxis se trouvant dans la liste $Liste_{RP}$ et dans lesquelles aucun arrêt est marqué par cette recherche, seront retirées de cette liste.

5.5.3 Destination vers les arrêts des taxis $P_{e\rightsquigarrow.}$

Une fois que le taxi a atteint la destination e du nouveau client c_r, il peut arriver que d'autres clients attendent toujours l'arrivée à leur destination ou bien leur prise en charge depuis leur lieu de départ. Dans ce cas, des plus courts chemins doivent être calculés à partir de la destination e vers les arrêts restants. Cela se fait avec un algorithme de Dijkstra borné, dans lequel nous limitons l'expansion jusqu'à ce qu'une durée maximale $D_r^{dest->arrets}$ soit atteinte.

$$D_r^{dest->arrets} = \max_{j\in P_z, \forall z\in Liste_{RP}} \left\{ t_{\max j} - (t_s^z + \delta(s,e)) \right\}$$

Où P_z est l'ensemble des arrêts des clients se trouvant dans le taxi z. Cette limite correspond à la situation dans laquelle un taxi z est à l'instant t_s^z à l'emplacement s. Il conduit jusqu'à la destination e puis poursuit sa route vers les autres arrêts qui sont les destinations/origines du reste de ses clients.

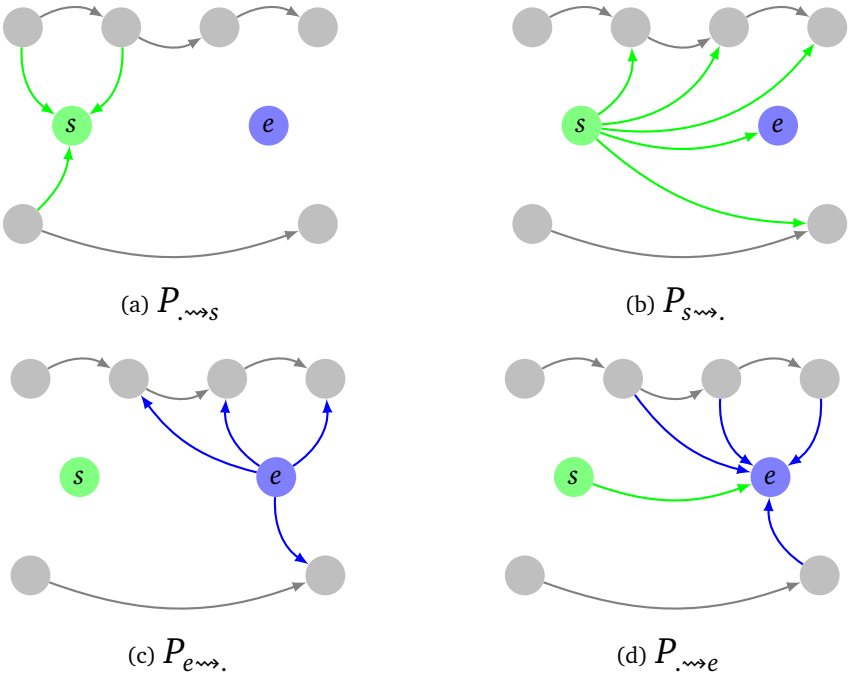
5.5.4 Arrêts des taxis vers la destination $P_{.\rightsquigarrow e}$

Nous supposons qu'un taxi prend en charge un nouveau client c_r, puis se dirige vers des arrêts qui sont les destinations/origines de certains de ses autres clients. Par conséquent, des plus courts chemins doivent être calculés à partir de ces arrêts vers la destination e. Le calcul de ces plus courts chemins est fait à l'aide d'un algorithme de Dijkstra inverse sur le graphe borné. La borne $D_r^{arrets->dest}$ est fixée à

$$D_r^{arrets->dest} = \max_{j\in P_z, \forall z\in Liste_{RP}} \left\{ D_r^{dest} - \delta(s,c_j) \right\}$$

5.5.5 Insertion d'un nouveau client

Jusqu'à présent, quatre ensembles de plus courts chemins ont été calculés. Une représentation schématique de la situation est illustrée dans la figure 5.3. La demande du nouveau client est représentée par un nœud vert pour le lieu de prise en charge et par un nœud bleu pour sa destination. Les autres nœuds représentent soit l'emplacement actuel des taxis, soit les arrêts (origines ou destinations) des clients. Dans la figure 5.3 il y a deux taxis avec leur itinéraire : le premier contient quatre nœuds et le second contient deux nœuds.



(a) $P_{.\rightsquigarrow s}$ (b) $P_{s\rightsquigarrow .}$

(c) $P_{e\rightsquigarrow .}$ (d) $P_{.\rightsquigarrow e}$

FIGURE 5.3 – Les quatres étapes constituant la première approche.

Les plus courts chemins des taxis de leurs arrêts/positions vers le nouveau lieu de prise en charge s se situent dans la partie supérieure gauche, le lieu de prise en charge du nouveau client vers sa destination en haut à droite. De la nouvelle destination vers les arrêts en bas à gauche, et des arrêts vers la destination en bas à droite. En effectuant quatre recherches de plus court chemin, deux recherches vers l'avant et deux recherches inverses, notre algorithme trouve tous les chemins permettant de construire une solution éventuelle *admissible* pour le problème de DDARPTW. Tous les itinéraires construits à partir de ces quatre ensembles de chemins ne sont pas forcément des solutions réalisables mais tous les itinéraires admissibles peuvent être construits à partir d'eux. Une vérification de faisabilité est donc effectuée sur la durée maximale du trajet ainsi que le coût associé à chaque passager. La capacité du véhicule est déjà vérifiée lors de la construction des routes admissibles. L'idée principale de cette heuristique d'insertion est la suivante : commencer par l'itinéraire du taxi, de son emplacement actuel x_0 vers ses différents arrêts (origine ou destination de ses passagers) $x_1, \ldots, x_{r-1}$. Dans cet itinéraire, nous essayons d'insérer le nouveau lieu de prise en charge s et la nouvelle destination e. Nous ne changeons pas l'ordre relatif $x_1, \ldots, x_{r-1}$ des arrêts, de sorte que le nombre de solutions soit raisonnable : r possibilités pour l'insertion du lieu de prise en charge s, et au plus r possibilités pour la destination e qui doit être insérée après s dans l'itinéraire. Le nombre de ces chemins qui comprennent s et e est $\frac{r(r+1)}{2}$.

Dans l'algorithme 11, nous considérons le point à insérer comme étant le lieu de prise en charge.
À partir d'un itinéraire P associé à un taxi, on essaye d'insérer un nouveau point s après la position i dans l'itinéraire P. Comme ce nouvel itinéraire pourrait être inadmissible, un premier test doit être effectué pour vérifier si le chemin le plus rapide de i vers s, puis de s vers $i+1$ est possible, c'est à dire qui respecte le délai d'attente maximum $maxwait$ et la capacité maximale du véhicule (étape 2). Si ce n'est pas le cas, alors l'algorithme renvoie une valeur nulle, sinon une mise à jour des heures d'arrivées des passagers du taxi se fait avec t' (étapes 9 et 20). Cela permet de vérifier le temps d'arrivée maximal $t_{\max j}$ à chaque destination (étapes 16 et 27) en déterminant le dernier arrêt dans lequel le nouveau client peut être déposé sans enfreindre la capacité du véhicule (étape 31).
Les coûts associés aux passagers du taxi ainsi que le coût du nouveau client sont calculés par les deux étapes 25 et 26. L'étape 30 assure l'attractivité des coûts aux passagers de type $< +, s, -, e >$. Ainsi, si chaque temps d'arrivée est respecté, alors le nouvel itinéraire P' est admissible et l'algorithme retourne P'. Sa complexité est en $O(r)$, où r est le nombre d'arrêts de l'actuel itinéraire P.
Basé sur le nombre d'occupants du véhicule à chaque arrêt, l'emplacement le plus éloigné possible dans lequel le client peut être déposé est déterminé par $Position\_max$. Une fois que le lieu de prise en charge a été inséré avec succès dans la route P, l'algorithme 12 est lancé pour chaque arrêt situé entre s et la $Position\_max$ afin d'insérer l'emplacement de dépose e du nouveau client. L'étape 1 l'algorithme 12 vérifie si les deux arrêts entre lesquels le lieu de dépose t doit être inséré (c'est à dire, v_i et v_{i+1}) ont été respectivement marqués par les deux étapes "Arrêts des taxis vers la destination" et "Destination vers les arrêts des taxis". En plus de la contrainte du temps d'arrivée (étapes 23 et 28), l'étape 8 de l'algorithme 12 vérifie la faisabilité du coût affecté au nouveau client ainsi que celui des passagers à bord du véhicule, ce qui correspond aux passagers de type $< +, s, e, - >$. Ce coût doit être respectivement inférieur ou égal au coût fixé par le nouveau client et aux coûts déjà affectés aux passagers avant l'insertion du nouveau client. L'étape 18 permet d'assurer la faisabilité sur les nouveaux coûts associés aux passagers de type $< s, +, e, - >$. Dans le cas où le lieu de dépose t est inséré après le dernier arrêt v_{r-1}, la nouvelle route P' est admissible si et seulement le temps d'arrivée maximal du nouveau client est respecté (c'est à dire $t_r \leq t_{\max r}$).

Algorithme 11 Insertion d'un nouveau point de prise en charge

Entrée: Itinéraire P | Nb de passagers à l'arrêt l_j | L'heure d'arrivée t_j | L'heure d'arrivée au plus tard à j $t_{\max j}$, $j = 0,\dots,r-1$ | $< i,s >$ la position de prise en charge s à insérer après l'arrêt i.

Sortie: Nouvel itinéraire P' avec les nouveaux temps d'arrivées t'| L'emplacement le plus éloigné possible de dépose *Position_max* | $C^{s,e}$ coût du nouveau client pour un trajet seul de s à e.
Coût_par_client(j), $j = i,\dots,r-1$ Coût par client du trajet depuis l'arrêt i jusqu'à l'arrêt j avant l'insertion du nouveau lieu de prise en charge s.
Coût_nouveau_client(j), $j = i,\dots,r-1$ Coût du nouveau client pour le trajet depuis son lieu de départ s jusqu'à l'arrêt j.

Initialisation : *Position_trouvée* ← Faux, *Coût_réduit* ← 0, *PriseEnCharge_entre_s_e*← ∅
Si $v_i \rightsquigarrow s \notin P_{.\rightsquigarrow s}$ **ou** $s \rightsquigarrow v_{i+1} \notin P_{s\rightsquigarrow .}$ **ou** $l_{v_i} = L$ **Alors**
  Retourner null
Fin Si
$P' \leftarrow$ Insérer s après v_i dans P
Pour tout $j = 0,\dots,i$ **Faire**
  $t'_j \leftarrow t_j, \quad l'_j \leftarrow l_j$
Fin Pour
$t'_s \leftarrow t_i + \delta(v_i,s), \quad t'_{i+1} \leftarrow t'_s + \delta(s,v_{i+1})$
$l'_s \leftarrow l_i + 1, \quad l'_{i+1} \leftarrow l_{i+1} + 1.$
Position_max ← $r-1$
Coût_supplémentaire ← $\max\left(\frac{C(i,s)}{l_i} + \frac{C(s,i+1)}{l_i+1} - \frac{C(i,i+1)}{l_i}, 0\right)$
Coût_réduit ← $\max\left(\frac{C(i,i+1)}{l_i} - (\frac{C(i,s)}{l_i} + \frac{C(s,i+1)}{l_i+1}), 0\right)$
Coût_nouveau_client(i+1) ← $\frac{C(s,i+1)}{l'_{i+1}}$.
Coût_par_client(i+1)← $\frac{C(i,i+1)}{l_{i+1}}$
Si $t'_s + \delta(s,e) > t_{\max s}$ **ou** $t'_{i+1} > t_{\max i+1}$ **Alors**
  Retourner null
Fin Si
Pour tout $j = i+2,\dots,r-1$ **Faire**
  $t'_j \leftarrow t'_{j-1} + (t_j - t_{j-1}), \quad l'_j \leftarrow l_j + 1$
  Si v_j est un arrêt de prise en charge **Alors**
    PriseEnCharge_entre_s_e ← *PriseEnCharge_entre_s_e* $\cup\{v_j\}$
  Fin Si
  Coût_réduit ← *Coût_réduit* $+\left(\frac{C(v_{j-1},v_j)}{l_j} - \frac{C(v_{j-1},v_j)}{l'_j}\right)$
  Coût_nouveau_client(j) ← $\left(\textit{Coût\_nouveau\_client(j)} + \frac{C(v_{j-1},v_j)}{l'_j}\right)$
  Coût_par_client(j) ← $\left(\textit{Coût\_par\_client(j)} + \frac{C(v_{j-1},v_j)}{l_j}\right)$
  Si $t'_j > t_{\max j}$ **Alors**
    Retourner null
  Fin Si
  Si $\left(l'_j > L$ **ou** *Coût_nouveau_client(j)* $\geq C^{s,e}$ **ou** (v_j correspond à un arrêt de dépose **et** l'arrêt de prise en charge associé à $v_j \notin$ *PriseEnCharge_entre_s_e* **et** *Coût_supplémentaire* > *Coût_réduit*$\right)$ **et** *Position_trouvée* = Faux **Alors**
    Position_max ← $j-1$, *Position_trouvée* ← Vrai.
  Fin Si
Fin Pour
Retourner P' et *Position_max*.

Algorithme 12 Insertion d'un nouveau point de dépose dans l'itinéraire P

Entrée: Itinéraire P | Nombre de passagers à l'arrêt l_j | L'heure d'arrivée à l'arrêt t_j | L'heure d'arrivée au plus tard à j $t_{\max j}$, $j = 0, \ldots, r-1$ | $< i, e >$ la position de dépose e à insérer après l'arrêt i | $< m, s >$ La position où le lieu de prise en charge a été inséré.
$Position\_max$ Emplacement le plus éloigné possible de dépose pour le passager courant.
Coût_par_client(j) Coût par client du trajet depuis l'arrêt m jusqu'à l'arrêt j avant l'insertion du nouveau lieu de prise en charge s.
Coût_nouveau_client(j), $j = i, \ldots, r-1$ Coût du nouveau client pour le trajet depuis son lieu de départ s jusqu'à l'arrêt i.

Sortie: Nouvel itinéraire P' avec les nouveaux temps d'arrivées t' ou échec d'insertion.

Si ($v_i \rightsquigarrow e \notin P_{\cdot \rightsquigarrow e}$ **et** $v_i \neq s$) **ou** $e \rightsquigarrow v_{i+1} \notin P_{e \rightsquigarrow \cdot}$ **Alors**
  Retourner null
Fin Si
$P' \leftarrow$ Insérer e après v_i dans P
Coût_réduit $\leftarrow 0$, $Depose\_entre\_s\_e \leftarrow \emptyset$
{Soit m l'arrêt se trouvant juste avant le nouveau lieu de prise en charge s dans P}
$Gain\_trajet \leftarrow$ *Coût_par_client(i+1)* $-\left(\frac{C(m,s)}{l_m} +$ *Coût_nouveau_client(i)* $+\frac{C(i,e)}{l_i} + \frac{C(e,i+1)}{l_{i+1}}\right)$
Gain_nouveau_client $\leftarrow C^{s,e} - \left($*Coût_nouveau_client(i)*$+\frac{C(i,e)}{l_i}\right)$
Si $Gain\_trajet < 0$ **ou** $Gain\_nouveau\_client < 0$ **Alors**
  Retourner nul
Fin Si
Coût_supplémentaire $= \max\left(\frac{C(i,e)}{l_i} + \frac{C(e,i+1)}{l_{i+1}-1} - \frac{C(i,i+1)}{l_{i+1}-1}, 0\right)$
Pour tout $j = v_i, \ldots, 0$ **Faire**
  $t'_j \leftarrow t_j$, $l'_j \leftarrow l_j$
  Si v_j est un arrêt de dépose **Alors**
    $Depose\_entre\_s\_e \leftarrow Depose\_entre\_s\_e \cup \{v_j\}$
  Fin Si
  Coût_réduit $=$ *Coût_réduit*$+ \left(\frac{C(v_j,v_{j+1})}{l_{j+1}-1} - \frac{C(v_j,v_{j+1})}{l_{j+1}}\right)$
  Si (v_j correspond à un arrêt de prise en charge **et** l'arrêt de dépose associé à $v_j \notin Depose\_entre\_s\_e$ **et** *Coût_supplémentaire* $>$ *Coût_réduit*) **Alors**
    Retourner null
  Fin Si
Fin Pour
$t'_t \leftarrow t_i + \delta(v_i, e)$, $t'_{i+1} \leftarrow t'_e + \delta(e, v_{i+1})$, $l'_e \leftarrow l_i - 1$, $l'_{i+1} \leftarrow l_{i+1} - 1$
Si $t'_e > t_{\max e}$ **ou** $t'_{i+1} > t_{\max i+1}$ **Alors**
  Retourner null
Fin Si
Pour tout $j = i+2, \ldots, r-1$ **Faire**
  $t'_j \leftarrow t'_{j-1} + (t_j - t_{j-1})$
  Si $t'_j > t_{\max j}$ **Alors**
    Retourner null
  Fin Si
  Si $j \leq Position\_max$ **Alors**
    $l'_j \leftarrow l_j - 1$
  Fin Si
Fin Pour
Retourner P'

L'algorithme 13 tente d'insérer le nouveau lieu de prise en charge s et la nouvelle destination e à l'intérieur de l'itinéraire actuel P d'un taxi. Il utilise les deux algorithmes 11 et 12 comme sous-programmes. Toutes les solutions ne sont pas testées car l'ordre relatif des arrêts est maintenu lors des tentatives d'insertion. Si les insertions de s et e génèrent une solution admissible, cette nouvelle solution pourrait être la meilleure solution en termes de *coût d'insertion relatif* et donc une mise à jour de cette dernière se fait par l'algorithme 14. Ainsi, la complexité de l'algorithme 13 dans le pire des cas est en $O(r^2)$ car il y a une boucle principale incluant une boucle interne qui fait appel aux deux algorithmes 11 et 12, où r est le nombre d'arrêts dans l'itinéraire actuel P. L'algorithme 13 teste également l'insertion du lieu de dépose t juste après le lieu de prise en charge s (c'est à dire, le cas $v_i \rightarrow s \rightarrow t \rightarrow v_{i+1}$) et à la fin de l'itinéraire dans le cas où *Position_max* correspond au dernier arrêt de la route (c'est à dire, le cas $* \rightarrow v_{r-1} \rightarrow t$). Dans ce dernier, le nouvel itinéraire P'' est admissible si et seulement si le temps d'arrivée maximal du nouveau client est respecté (c'est à dire $t_r \leq t_{\max r}$).

Algorithme 13 Insertion d'un nouveau client

Entrée: Itinéraire $P = v_0, v_1, \ldots, v_{r-1}$
s, e Nouvelles positions de prise en charge et de dépose à insérer.
Sortie: Meilleure insertion *P*best, Nombre de solutions *nbsolution* ou échec d'insertion.

nbsolution ← 0
Pour tout $i = 0, \ldots, r-1$ **Faire**
  Déterminer $(P', Position\_max)$ en insérant s après l'arrêt i en utilisant l'**algorithme 11**
  Si P' est non nul **Alors**
    Pour tout $j = s, i+1 \ldots Position\_max$ **Faire**
      Déterminer P'' en insérant e dans P' après l'arrêt j en utilisant l'**algorithme 12**
      Si P'' est non nul **Alors**
        nbsolution ← *nbsolution* +1
        UpdateBest(*P*best,P'',*nbsolution*)
      Sinon
        STOP
      Fin Si
    Fin Pour
  Fin Si
Fin Pour
Retourner *P*best

Algorithme 14 UpdateBest

Entrée: Itinéraire *P*best, P''
Nombre de solutions *nbsolution*
Sortie: Meilleur itinéraire *P*best
Si *nbsolution*=1 **Alors**
  *P*best ← P''
Sinon
  Si P'' est "meilleur" que *P*best en termes de la fonction objectif f_r **Alors**
    *P*best ← P''
  Fin Si
Fin Si
Retourner *P*best

L'algorithme 14 vérifie d'abord si c'est la première fois qu'une solution admissible P'' est soumise. Si cela est le cas, alors P'' est de facto la meilleure solution trouvée jusqu'ici. Sinon, la valeur de la fonction objectif est comparée entre la meilleure solution courante $Pbest$ et la nouvelle solution P''. Sa complexité est en $O(1)$.

5.5.6 Arrêts intermédiaires de prise en charge et de dépose des clients

Dans cette section, nous étendons notre modèle pour offrir aux clients la possibilité de marcher sur une courte distance (où une courte durée fixée au préalable) à partir de leur lieu de départ vers un arrêt de prise en charge et/ou à partir d'un arrêt de dépose vers leur destination. Les arrêts possibles de prise en charge et de dépose correspondent aux positions et aux arrêts des taxis. Cela permet d'assurer la continuité du taxi sur son itinéraire actuel de la manière la plus efficace possible. D'autre part, cela permet d'éviter les coûts supplémentaires survenus sur les détours des conducteurs.
L'algorithme 15 détermine les arrêts potentiels de prise en charge et de dépose en fonction des durées respectives *Durée_max_depuis_origine* et *Durée_max_vers_destination*.

Algorithme 15 Arrêts intermédiaires potentiels de prise en charge et de dépose pour le nouveau client

Entrée: Graphe G, lieu de départ s, destination e.
Durée_max_depuis_origine : durée maximale tolérée par le client pour rejoindre un arrêt intermédiaire de prise en charge depuis son lieu de départ.
Durée_max_vers_destination : durée maximale tolérée par le client pour rejoindre sa destination depuis un arrêt intermédiaire de dépose.

Sortie: *Liste_PriseEnCharge_Intermédiaires* : liste des arrêts des taxis atteignables par le client r en marche à pied depuis son origine s en respectant la durée maximale *Durée_max_depuis_origine* et la capacité du véhicule.
Liste_Dépose_Intermédiaires : liste des arrêts des taxis atteignables par le client r dans lesquels il pourrait atteindre à pied sa destination e en respectant la durée maximale et la capacité du véhicule.

1: **Initialisation :** *Liste_PriseEnCharge_Intermédiaires* $\leftarrow \emptyset$
Liste_Dépose_Intermédiaires $\leftarrow \emptyset$
2: Calculer un algorithme de Dijkstra depuis s limité par la durée *Durée_max_depuis_origine*
3: **Pour tout** noeud marqué v correspondant à une position actuelle **ou** un arrêt d'un taxi z **Faire**
4: **Si** la capacité maximale du taxi à l'arrêt v n'est pas atteinte **et** $(t_s + \delta_{walk}(s,v) \leq t_{\max v})$ **Alors**
5: *Liste_PriseEnCharge_Intermédiaires* $\leftarrow$ *Liste_PriseEnCharge_Intermédiaires* $\cup \{v\}$.
6: **Fin Si**
7: **Fin Pour**
8: Calculer un algorithme de Dijkstra inverse depuis e limité par la durée *Durée_max_vers_destination*.
9: **Pour tout** noeud marqué v correspondant à l'arrêt d'un taxi **Faire**
10: **Si** la capacité maximale du taxi à l'arrêt v n'est pas atteinte **et** $(t_v + \delta_{walk}(v,e) \leq t_{\max e})$ **Alors**
11: *Liste_Dépose_Intermédiaires* $\leftarrow$ *Liste_Dépose_Intermédiaires* $\cup \{v\}$.
12: **Fin Si**
13: **Fin Pour**

L'insertion du point de prise en charge s du nouveau client c_r dans la route d'un taxi (après l'arrêt v_i) dépend de l'appartenance de l'arrêt v_i à $Liste_{AP}$ (déterminée par l'algorithme 10) et de l'arrêt v_{i+1} à *Liste_PriseEnCharge_Intermédiaires*. En effet, lorsqu'une des deux conditions n'est pas

respectée (c'est à dire, soit $v_i \notin Liste_{AP}$ ou $v_{i+1} \notin$ *Liste_PriseEnCharge_Intermédiaires*), l'insertion du lieu de prise en charge s après l'arrêt v_i se fait sans aucun chevauchement entre les deux possibilités (voir l'algorithme 16). Dans un premier temps, l'algorithme 16 teste si l'arrêt v_{i+1} est atteignable par le client en marche à pied. Si ce n'est pas le cas, l'insertion du nouveau lieu de prise en charge s se fait directement avec l'algorithme 11. Autrement, l'insertion du nouveau lieu de prise en charge s est définie de l'étape 6 à 27 en déterminant également l'emplacement le plus éloigné possible pour la dépose du nouveau client.

Algorithme 16 Insertion d'un nouveau lieu de prise en charge après l'arrêt v_i lorsque $v_i \notin Liste_{AP}$ ou $v_{i+1} \notin$ *Liste_Prise-en-charge_Intermédiaire*

Entrée: *Liste_Prise-en-charge_Intermédiaires* | $Liste_{AP}$ | s lieu de prise en charge
Itinéraire $P = v_0, v_1, \ldots, v_{r-1}$ avec le nombre de passagers à bord du véhicule à chaque $l = l_0, l_1, \ldots, l_{r-1}$.
$t_j, j = 0, \ldots, k-1$ l'heure d'arrivée à chaque arrêt v_j.
$t_{\max j}, j = 0, \ldots, r-1$ l'heure d'arrivée maximale à chaque arrêt v_j.
Sortie: Nouvel itinéraire P' avec les nouveaux temps d'arrivées t' et la position la plus éloignée $Position\_max$ pour l'insertion du point de dépose e.
Coût_par_client(j) Coût par client du trajet depuis l'arrêt i jusqu'à l'arrêt j avant l'insertion du nouveau lieu de prise en charge s.
Coût_nouveau_client(j), $j = i \ldots r-1$ Coût du nouveau client pour le trajet depuis son lieu de départ s jusqu'à l'arrêt j.

Initialisation : position_trouvée ← Faux, $Position\_max \leftarrow 0$.
Si $v_i \in Liste_{AP}$ **et** $v_{i+1} \notin$ *Liste_PriseEnCharge_Intermédiaires* **Alors**
  Retourner $P', Position\_max$ en utilisant l'**algorithme 11**
Fin Si
Si $v_i \notin Liste_{AP}$ **et** $v_{i+1} \in$ *Liste_PriseEnCharge_Intermédiaires* **Alors**
  Pour tout $j = 0, \ldots, i$ **Faire**
    $t'_j \leftarrow t_j$, $l'_j \leftarrow l_j$
  Fin Pour
  $t'_{i+1} \leftarrow \max\{t_{i+1}, t_s + \delta_{walk}(s, v_{i+1})\}$, $l'_s \leftarrow l'_{i+1} \leftarrow l_{i+1} + 1$
  Coût_nouveau_client(i+1) ← 0.
  Coût_par_client(i+1) ← 0
  Si $t'_{i+1} > t_{\max i+1}$ **Alors**
    Retourner null
  Fin Si
  Pour tout $j = i+2, \ldots, r-1$ **Faire**
    $t'_j \leftarrow t'_{j-1} + (t_j - t_{j-1})$, $l'_j \leftarrow l_j + 1$
    Coût_nouveau_client(j) ← $\left(\textit{Coût\_nouveau\_client(j-1)} + \frac{C(v_{j-1}, v_j)}{l'_j}\right)$
    Coût_par_client(j) ← $\left(\textit{Coût\_par\_client(j-1)} + \frac{C(v_{j-1}, v_j)}{l_j}\right)$
    Si $t'_j > t_{\max j}$ **Alors**
      Retourner null
    Fin Si
    Si $l'_j > L$ **et** position_trouvée = Faux **Alors**
      $Position\_max \leftarrow j-1$
      position_trouvée ← Vrai
    Fin Si
  Fin Pour
  Retourner $P', Position\_max$
Fin Si
Retourner null

L'algorithme 17 décrit l'insertion du point de prise en charge s du nouveau client c_r après l'arrêt v_i lorsque v_i et v_{i+1} appartiennent simultanément à $Liste_{AP}$ et *Liste_PriseEnCharge_Intermédiaires* Lorsque la condition de l'étape 3 n'est pas vérifiée, deux nouvelles routes potentielles $P_1^{'}$ et $P_2^{'}$ sont construites. Aucune des deux ne domine l'autre en termes de coût d'insertion et de temps d'arrivée à v_{i+1}, d'où la nécessité de garder les deux routes. La figure 5.4 illustre ce procédé. Comme on peut le constater d'après cette dernière, $P_2^{'}$ est plus attractif en termes de coût d'insertion (égal à 0) et $P_1^{'}$ est plus attractif en termes de temps d'arrivée à v_{i+1} (qui correspond à l'instant 9h04). Ainsi, nous ne pouvons pas choisir $P_2^{'}$ malgré son attractivité en coût d'insertion.

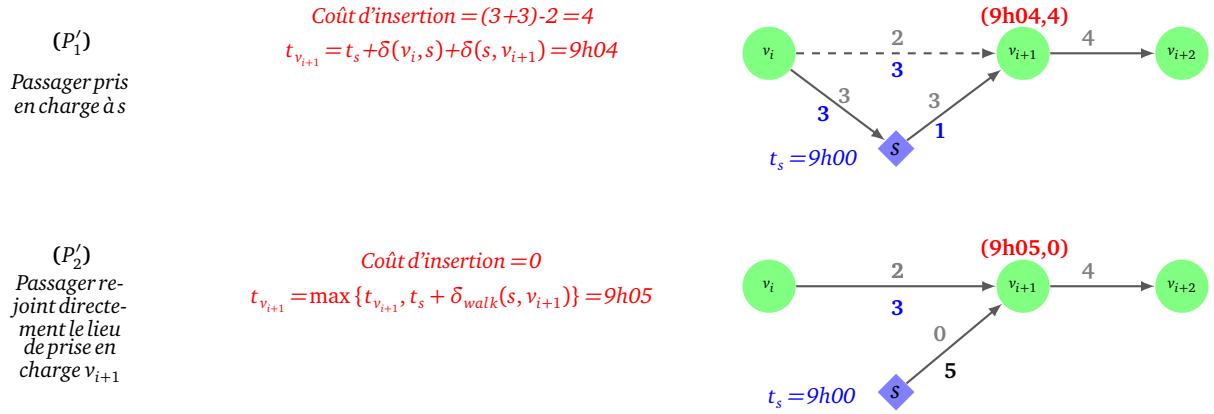


FIGURE 5.4 – Les deux insertions possibles après l'arrêt v_i lorsque ce dernier appartient à la liste $Liste_{AP}$ et que v_{i+1} appartient à la liste *Liste_PriseEnCharge_Intermédiaires* sachant que l'étape 3 de l'algorithme 17 n'est pas respectée. Les valeurs de couleur bleue correspondent aux durées nécessaires pour traverser le segment en voiture, tandis que la valeur de couleur noir correspond à la durée nécessaire en marche à pied. Concernant les coûts des chemins, ces derniers sont représentés en gris. Par exemple, le coût du chemin depuis s vers v_{i+1} sur la sous-figure (P_2') est nul car le client rejoint par ses propores moyen l'arrêt v_{i+1}. En revanche dans la sous-figure (P_1'), les coûts de déplacement en voiture pour la prise en charge du passager sont respectivement $v_i \rightsquigarrow s = 3$ et de $s \rightsquigarrow v_{i+1} = 1$.

Ainsi, lorsque l'étape 3 de l'algorithme 17 est vérifiée, la route $P_2^{'}$ domine $P_1^{'}$.

Algorithme 17 Insertion d'un nouveau lieu de prise en charge après l'arrêt v_i lorsque $v_i \in Liste_{AP}$ et $v_{i+1} \in Liste\_PriseEnCharge\_Intermédiaires$

Entrée: $Liste\_PriseEnCharge\_Intermédiaires$ | $Liste_{AP}$ | s lieu de prise en charge | Itinéraire $P = v_0, v_1, \ldots, v_{r-1}$ avec le nombre de passagers à bord du véhicule à chaque $l = l_0, l_1, \ldots, l_{r-1}$.
$t_j, j = 0, \ldots, r-1$ l'heure d'arrivée à chaque arrêt v_j.
$t_{\max j}, j = 0, \ldots, r-1$ l'heure d'arrivée au plus tard à chaque arrêt v_j.

Sortie: Nouvel itinéraire P_1' avec les nouveaux temps d'arrivées t' et la position la plus éloignée $Position\_max$ ou null.
Nouvel itinéraire P_2' avec les nouveaux temps d'arrivées t'' et la position la plus éloignée $Position\_max\_walk$ ou null.

1: **Initialisation :** Position_trouvée ← Faux, $Position\_max$ ← -1, $Position\_max\_walk$ ← -1.
2: **Si** $v_i \in Liste_{AP}$ **et** $v_{i+1} \in Liste\_PriseEnCharge\_Intermédiaires$ **Alors**
3: **Si** $\max\{t_{i+1}, t_s + \delta_{walk}(s, v_{i+1})\} < (t_i + \delta(v_i, s) + \delta(s, v_{i+1}))$ **Alors**
4: Retourner P_2', $Position\_max\_walk$ en utilisant l'**algorithme 16** de l'étape **6** à l'étape **27**
5: **Sinon**
6: Déterminer P_1' et $Position\_max$ en utilisant l'**algorithme 11**
7: Déterminer P_2' et $Position\_max\_walk$ en utilisant l'**algorithme 16** de l'étape **6** à l'étape **27**
8: Retourner P_1' et P_2'
9: **Fin Si**
10: **Fin Si**
11: Retourner null

Afin d'insérer le lieu de dépose du nouveau client c_r, l'algorithme 18 est exécuté pour chaque arrêt situé entre son point de prise en charge et le dernier arrêt possible de dépose (c'est à dire, le maximum entre $Position\_max\_walk$ et $Position\_max$). Ce dernier est basé sur le même raisonnement que celui de l'insertion du lieu de prise en charge s décrit dans l'algorithme 17. Plus exactement, l'algorithme 18 vérifie tout d'abord si l'arrêt $v_i \in Liste\_Dépose\_Intermédiaires$ et que le lieu de dépose e peut potentiellement être inséré entre v_i et v_{i+1}. Si c'est le cas, l'étape 5 vérifie l'admissibilité du temps d'arrivée maximal du nouveau client lorsqu'il va rejoindre directement sa destination à partir de l'arrêt v_i. Si cela est vérifiée, l'insertion du nouveau lieu de prise en charge entre v_i et v_{i+1} n'est plus nécessaire car le coût d'insertion ne peut que croître.

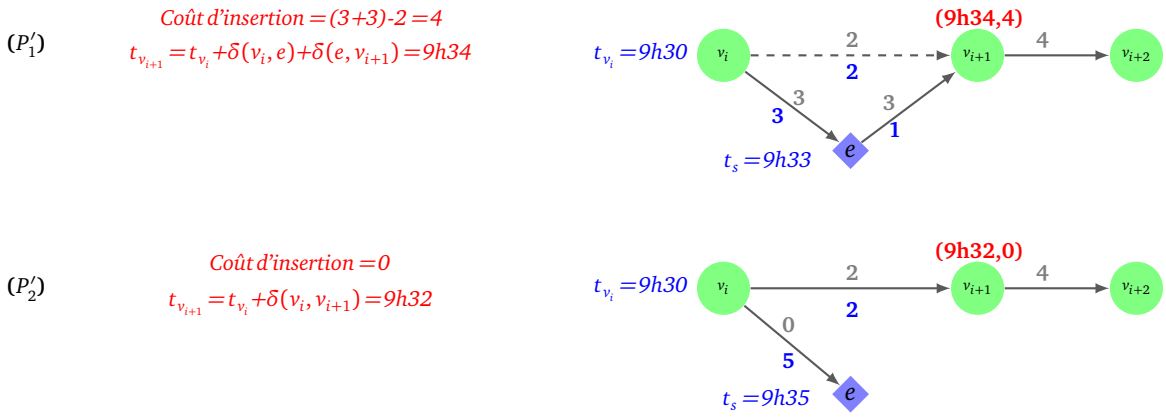


FIGURE 5.5 – Les deux insertions possibles après l'arrêt v_i lorsque le lieu de dépose e peut potentiellement être inséré entre v_i et v_{i+1} et que v_i appartient à la liste *Liste_Dépose_Intermédiaires* sachant que l'étape 5 de l'algorithme 18 est vérifiée.

Comme on peut le constater d'après la figure 5.5, lorsque le nouveau client peut rejoindre sa destination à partir de l'arrêt v_i sans enfreindre son temps maximal d'arrivée, il n'est pas nécessaire d'essayer d'insérer e entre v_i et v_{i+1}. Dans ce cas, la route P_1' sera toujours dominée par P_2' en termes de coût d'insertion et de temps d'arrivée des autres passagers.

Algorithme 18 Insertion du point de dépose après l'arrêt v_i

Entrée: Itinéraire $P_1 = v_0, v_1, \ldots, v_{r-1}$ avec le nombre de passagers à bord du véhicule à chaque arrêt $l = l_0, l_1, \ldots, l_{r-1}$.
$t_j, j = 0, \ldots, r-1$ l'heure d'arrivée à chaque arrêt v_j
$t_{\max j}, j = 0, \ldots, r-1$ l'heure d'arrivée au plus tard à l'arrêt v_j, $j \in \{0, \ldots, r-1\}$ position d'insertion
e lieu de dépose du nouveau client.
La position la plus éloignée possible pour l'insertion du lieu de dépose $Position\_max\_walk$ et $Position\_max$.
$C^{s,e}$ coût du nouveau client pour un trajet en solo à partir de s à e.
Sortie: Nouvel itinéraire P' avec les nouveaux temps d'arrivées t'.

Si $\big((v_i \neq s$ **et** $v_i \rightsquigarrow e \notin P_{.\rightsquigarrow e})$ **ou** $e \rightsquigarrow v_{i+1} \notin P_{e\rightsquigarrow .}\big)$ **et** $v_i \notin$ *Liste_Dépose_Intermédiaires* **Alors**
  Retourner null
Fin Si
Si $(v_i = s$ **ou** $v_i \rightsquigarrow e \in P_{.\rightsquigarrow e})$ **et** $e \rightsquigarrow v_{i+1} \in P_{e\rightsquigarrow .}$ **Alors**
  Si $v_i \in$ *Liste_Dépose_Intermédiaires* **et** $(t_i + \delta_{walk}(v_i, e)) \leq t_{\max e}$ **Alors**
    Retourner P'
  Sinon
    Insérer le nouveau point de dépose e après v_i en utilisant l'**algorithme 12**
  Fin Si
Fin Si
Si $\big((v_i \neq s$ **et** $v_i \rightsquigarrow e \notin P_{.\rightsquigarrow e})$ **ou** $e \rightsquigarrow v_{i+1} \notin P_{e\rightsquigarrow .}\big)$ **et** $v_i \in$ *Liste_Dépose_Intermédiaires* **Alors**
  Si $(t_i + \delta_{walk}(v_i, e)) \leq t_{\max e}$ **Alors**
    Retourner P'
  Fin Si
Fin Si
Retourner null

L'algorithme 19 décrit les étapes nécessaires pour tester l'insertion du nouveau client c_r dans une route d'un taxi tout en prenant en compte les arrêts intermédiaires de prise en charge et de dépose. Il utilise les trois algorithmes 16, 17, et 18 comme des sous-programmes.

Algorithme 19 Insertion du nouveau client

Entrée: Itinéraire $P = v_0, v_1, \ldots, v_{r-1}$
s, e nouvelles positions de prise en charge et de dépose.
Sortie: Meilleure insertion Pbest ou échec d'insertion.

initialisation : Pbest $= null$.
Pour tout $i = 0, \ldots, r-1$ **Faire**
  $Position\_max \leftarrow 0, Position\_max\_walk \leftarrow 0$, $P'_1 \leftarrow null, P'_2 \leftarrow null$
  Si ($v_i \in Liste_{AP}$ **ou** $v_{i+1} \in$ *Liste_PriseEnCharge_Intermédiaires*) **et** non les deux **Alors**
    Déterminer $(P'_1, Position\_max)$ en insérant s dans P après l'arrêt v_i en utilisant l'**algorithme 16**
  Fin Si
  Si $v_i \in Liste_{AP}$ **et** $v_{i+1} \in$ *Liste_PriseEnCharge_Intermédiaires* **Alors**
    Déterminer $(P'_1, P'_2, Position\_max, Position\_max\_walk)$ en insérant s dans P après l'arrêt v_i en utilisant l'**algorithme 17**
  Fin Si
  Si P'_1 **ou** P'_2 est non null **Alors**
    Pour tout $j = s, i+1, \ldots, \max\{Position\_max, Position\_max\_walk\}$ **Faire**
      $P'' \leftarrow$ meilleure insertion du lieu de dépose e parmi les deux routes P'_1 et P'_2 en utilisant l'**algorithme 18**
      Si P'' est meilleur que Pbest **Alors**
        Mettre à jour $Pbest$ à P''
      Fin Si
    Fin Pour
  Fin Si
Fin Pour
Retourner $Pbest$.

5.5.7 Complexité de l'approche

L'étape de "Prise en charge" (section 5.5.1) qui permet de sélectionner les taxis potentiels peut être déterminée en utilisant un seul algorithme de Dijkstra limité par la durée $maxwait$. Les trois étapes, à savoir : "Origine de la demande vers sa destination et les arrêts des taxis" (section 5.5.2), "Destination vers les arrêts des taxis" (section 5.5.3) et "Arrêts des taxis vers la destination" (section 5.5.4) permettent de trouver toutes les durées nécessaires. Plus spécifiquement, les plus courtes durées depuis le lieu de départ de la requête vers les arrêts des taxis potentielles détectées dans l'étape précédente ainsi que les plus courtes durées depuis la destination de la nouvelle requête vers les arrêts de taxis potentiels. A ce niveau, toutes les durées nécessaires pour tester l'insertion de la nouvelle requête dans les routes des taxis potentiels sont déterminées. L'insertion d'un nouveau lieu de prise en charge suivi du lieu de dépose s'effectue en ${nb_k}^2$, où nb_k est le nombre d'arrêts constituant l'itinéraire d'un taxi k. Ainsi, la complexité de l'approche est de $O(4 \cdot Dijkstra + \sum_{k \in Liste_{RP}} O({nb_k}^2))$. La prise en compte de lieux intermédiaires de prise en charge et de dépose augmente légèrement la complexité de l'approche en utilisant deux autres algorithmes de Dijkstra. Les étapes restantes sur l'insertion du lieu de départ et de la destination de la nouvelle requête restent assez similaires à l'approche précédente (c'est à dire, sans la prise en compte des arrêts intermédiaires).

5.6 Deuxième approche en considérant une flotte mixte de taxis et de covoiturage

Dans la deuxième approche, nous nous positionnons dans un cadre plus général où les conducteurs particuliers peuvent également publier leurs trajets en temps réel. Dans ce cas, la flotte disponible est composée de conducteurs de taxis et de conducteurs particuliers. L'approche est composée de trois étapes. La première consiste à déterminer les conducteurs potentiels pour une prise en charge du nouveau client soit directement à partir de leur position actuelle soit en passant d'abord par les arrêts de leurs passagers. Après la détermination de ces derniers, la seconde étape consiste à insérer le point de prise en charge dans les différentes routes potentielles des taxis et des conducteurs particuliers. Pendant la procédure d'insertion du lieu de prise en charge, certaines bornes sont calculées. Ces dernières permettent d'insérer efficacement le lieu de dépose en limitant l'espace de recherche parcouru dans la troisième étape.
En considérant une flotte mixte composée de taxis et covoiturage, on retrouve deux types de trajectoires. Dans la trajectoire d'un covoiturage, le dernier arrêt correspond à la destination du conducteur, ce qui n'est pas le cas pour la trajectoire d'un taxi. Cela induit deux contraintes supplémentaires à considérer. Ces dernières concernent le coût et le temps de détour du conducteur particulier. La figure 5.6 représente les deux types de trajectoires, la première correspond à la trajectoire d'un taxi, et la deuxième correspond à la trajectoire d'un covoiturage.

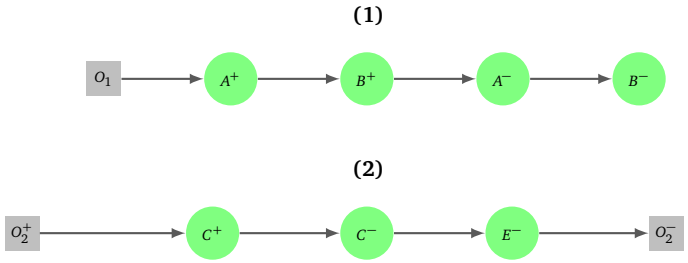


FIGURE 5.6 – La sous-figure (1) représente la trajectoire d'un taxi dont la position actuelle est O_1. En revanche, la sous-figure (2) représente la trajectoire d'un conducteur de covoiturage dont O_2^+ est sa postition actuelle et O_2^- est sa destination finale.

5.6.1 Prise en charge

La première étape est similaire à celle décrite dans 5.5.1. Cependant, contrairement à un conducteur de taxi, un conducteur particulier est limité par son temps de détour maximal. Ainsi, pour une position actuelle d'un conducteur particulier ainsi que les arrêts de ses passages, nous devons vérifier la faisabilité de la borne inférieure de la contrainte du temps de détour du conducteur avant de l'insérer dans la liste des routes potentielles $Liste_{RP}$ et la liste potentielle des arrêts de prise en charge $Liste_{AP}$. Cette contrainte garantit au conducteur un temps de détour en passant par le lieu de prise en charge s inférieur à son temps de détour maximal. La seule modification à effectuer dans l'algorithme 10 se trouve à l'étape 9. Plus exactement, lorsqu'un nœud v marqué par la recherche correspond à un des arrêts d'un conducteur privé z, nous devons vérifier la contrainte du temps de détour du conducteur passant par le nœud v. Le figure 5.7 représente la situation lorsqu'une requête d'un nouveau client arrive dans le système. Plus exactement, la stratégie de sélection des deux conducteurs potentiels ainsi que leurs arrêts se trouvent dans une zone géographique centrée sur le point de départ du client.

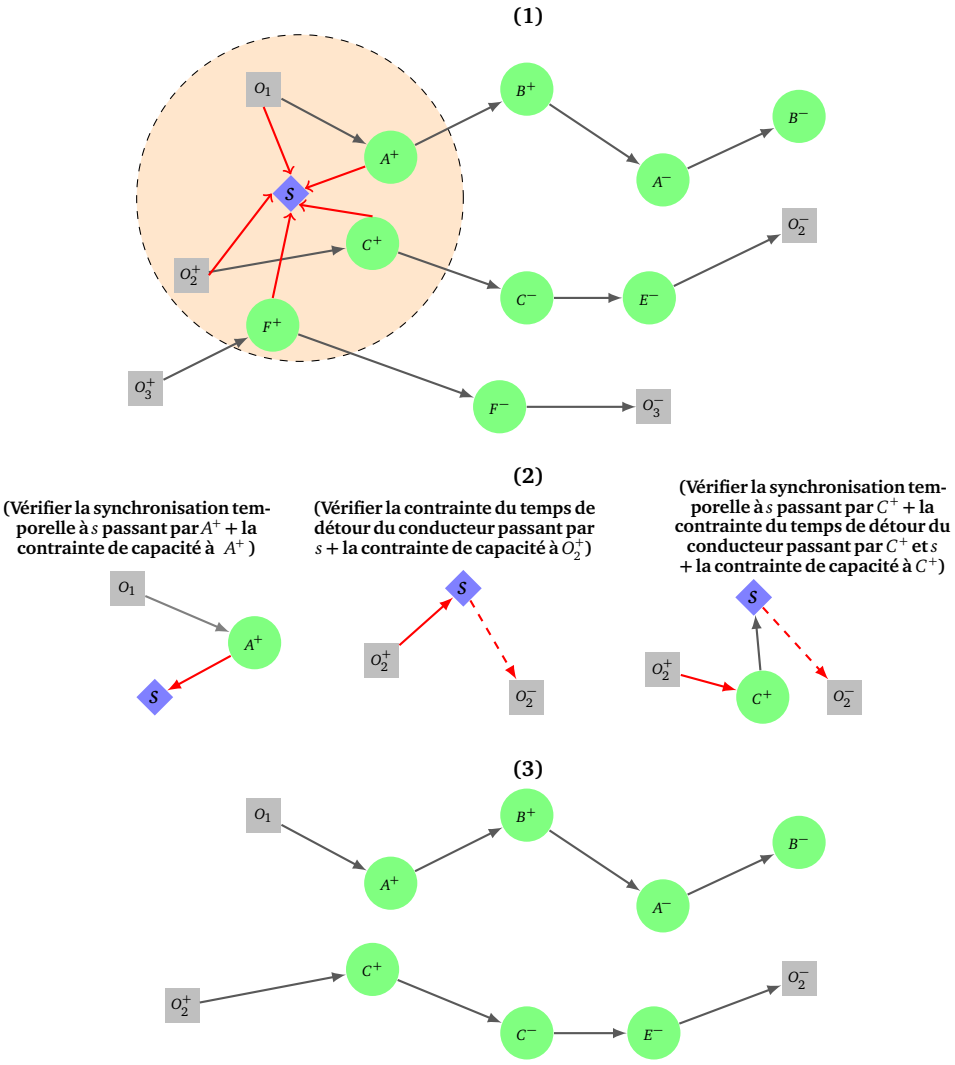


FIGURE 5.7 – La sous-figure (1) regroupe les conducteurs potentiels ainsi que leurs arrêts situés dans une zone géographique centrée sur le lieu de prise en charge s du nouveau client. Les positions actuelles des taxis dans lesquels la capacité maximale du véhicule n'est pas atteinte sont directement insérées dans la liste des conducteurs potentiels $\{O_1\}$. Avant d'insérer les deux arrêts $\{A^+, C^+\}$ dans la liste potentielle de prise en charge, nous devons vérifier la synchronisation temporelle entre le conducteur et le nouveau client au point de prise en charge s passant par ces deux arrêts. En plus des deux contraintes : la synchronisation temporelle et la contrainte de capacité du véhicule, la borne inférieure sur la contrainte du temps de détour des conducteurs privés doit être vérifiée pour les deux arrêts $\{O_2^+, C^+\}$ (sous-figure (2)). Les lignes en pointillé correspondent aux durées estimées en utilisant la formule Haversine. L'arrêt F^+ est directement éliminé car la position actuelle du conducteur associé n'est pas détectée dans le $maxwait$. Enfin, les deux itinéraires possibles sont présentés dans la sous-figure (3).

5.6.2 Insertion du lieu de prise en charge du nouveau client dans les routes potentielles

A ce niveau, nous avons déterminé les conducteurs potentiels pour une éventuelle prise en charge d'un nouveau client tout en respectant son temps d'attente maximal. A présent, nous allons insérer le nouveau point de prise en charge s dans les routes potentielles $Liste_{RP}$. Comme indiqué dans l'approche précédente, nous ne changeons pas l'ordre relatif des arrêts après l'insertion du nouveau point de prise en charge ou de dépose. Pour ce faire, nous utilisons l'algorithme de Dijkstra bidirectionnel A* modifié (BA). L'expansion de la recherche s'arrête lorsque la durée fixée à $D_r^{dest} = \lambda_r \delta(s,e)$ est atteinte. L'heure d'arrivée la plus tardive est fixée à

$$t_{\max r} = t_0 + maxwait + D_r^{dest}$$

où t_0 est le moment où la demande a été envoyée par le client c_r. Notons qu'une fois que le client r est affecté à un taxi z, nous mettons à jour son heure d'arrivée la plus tardive $t_{\max r}$ à $t_s^z + D_r^{dest}$.
Pendant la procédure de recherche, lorsqu'un nœud v_j correspondant à un des passagers à bord des conducteurs potentiels (détecté par l'algorithme 10) est marqué par les deux files de priorité, deux tests doivent être effectués. Premièrement, on doit vérifier si le prédécesseur du nœud marqué v_j (c'est à dire v_{j-1}) appartient à la liste $List_{AP}$. Cela permet de savoir si le conducteur est capable de prendre en charge le client de s directement après l'arrêt v_{j-1}. Si cela est vérifié, le second test vérifie l'admissibilité de l'heure maximale d'arrivée à chaque arrêt situé après v_j. Enfin, si les deux précédentes contraintes sont satisfaites, nous stockons l'information suivante :

(i) Coût d'insertion en termes de durée (c'est à dire $\delta(v_{j-1},s) + \delta(s,v_j) - \delta(v_{j-1},v_j)$) dans l'étiquette $Meilleure\_PriseEnCharge(v_j)$.

D'autres informations importantes à stocker concernent les successeurs des noeuds v_j marqués par les deux recherches (c'est à dire les v_{j+1}), dans lesquels au moins un des arrêts de leur trajet appartient à la liste $List_{AP}$. Ces derniers sont stockés dans la liste $Liste\_Potentielles\_PD$.
Ainsi, la liste *Liste_Potentielles_PD* contient les lieux des passagers qui peuvent être pris en charge ou déposés juste après avoir déposé le nouveau client. Une fois que l'algorithme BA est stoppé, nous mettons à jour la liste des routes potentielles $Liste_{RP}$ en supprimant tous les itinéraires dans lesquels aucun arrêt n'a été marqué par les deux recherches. Les différentes étapes sont décrites dans l'algorithme 20. La file de priorité de la *recherche avant* de Dijkstra bidirectionnel est représentée par Q_F, tandis que Q_B représente la file de priorité de la *recherche arrière*.
La figure 5.8 illustre le fonctionnement de ce procédé lorsqu'un arrêt est visité par les deux recherches.

Algorithme 20 Insertion du lieu de prise en charge du nouveau client dans les routes potentielles

Entrée: Graphe G, lieu de prise en charge s, destination e, coefficient de détour λ_r
$Liste_{AP}$: liste des arrêts et des positions potentielles des conducteurs
$Liste_{RP}$: liste des routes potentielles.
$RD_j = \min\{t_{\max i} - t_i, i = j, \ldots, r-1\}$ la plus petite durée restante avant de dépasser le temps maximal d'arrivée de chaque passager actuelle i situé après l'arrêt j, $0 \leq j \leq r-1$.
Nombre de passagers à bord du véhicule à chaque arrêt $l = l_0, l_1, \ldots, l_{r-1}$.

Sortie: $Meilleure\_PriseEnCharge(j)$: meilleure insertion possible du point de prise en charge avant l'arrêt j, $\forall j \in Liste_{RP}$
Liste_Arrêts_Potentiels : liste potentielle des arrêts de prise en charge et de dépose
Mise à jour de la liste des routes potentielles $Liste_{RP} \mid Charge\_max(v_j)$, $\forall v_j \in Liste_{RP}$.
Liste_Potentielles_PD : arrêts des passagers qui peuvent être pris en charge ou déposés juste après avoir déposé le nouveau client.

Initialisation : Q_F.*Insérer*(s), Q_B.*Insérer*(e), $D_r^{dest} \leftarrow \infty$, Recherche $\leftarrow$ Faux, $\delta(s,e) \leftarrow \infty$, *Liste_Arrêts_Potentiels* $\leftarrow \emptyset$
Tant que Recherche = Vrai **Faire**
  $v_j \leftarrow \min(Q_F.\min(), Q_B.\min())$
  Marquer le noeud v_j comme visité et retirer ce dernier de la file de priorité appropriée
  Si v_j est marqué visité par les deux files de priorité Q_F et Q_B **Alors**
    Si $v_j \in Liste_{RP}$ **et** v_j est un arrêt d'un conducteur **et** $v_{j-1} \in Liste_{AP}$ **Alors**
      Récupérer la route associée $P = v_0, \ldots, v_j, \ldots, v_{r-1}$ de la liste $Liste_{RP}$
      Si $\big(\delta(v_{j-1}, s) + \delta(s, v_j) - \delta(v_{j-1}, v_j)\big) \leq RD_j$ **et** $l_j + 1 \leq L$ **Alors**
        Liste_Arrêts_Potentiels $\leftarrow$ *Liste_Arrêts_Potentiels* $\cup \{v_j\}$
        $Meilleure\_PriseEnCharge(v_j) \leftarrow \{\delta(v_{j-1}, s) + \delta(s, v_j) - \delta(v_{j-1}, v_j)\}$
        $Charge\_max(v_j) \leftarrow$ Faux
      Sinon
        $Meilleure\_PriseEnCharge(v_j) \leftarrow -1$
        Si $l_j + 1 > L$ **Alors**
          $Charge\_max(v_j) \leftarrow$ Vrai
        Sinon
          $Charge\_max(v_j) \leftarrow$ Faux
        Fin Si
      Fin Si
      Liste_Potentielles_PD $\leftarrow$ *Liste_Potentielles_PD* $\cup \{v_{j+1}\}$
    Fin Si
    Si $\delta(s, v_j) + \delta(v_j, e) < \delta(s, e)$ **Alors**
      Mise à jour : $\delta(s,e) \leftarrow \delta(s, v_j) + \delta(v_j, e)$ et $D_k^{dest} \leftarrow \lambda_k \delta(s,e)$
    Fin Si
    Si $\min\big\{\delta(s, v_j), \delta(v_j, e)\big\} > D_k^{dest}$ **Alors**
      Recherche $\leftarrow$ Faux
    Fin Si
  Fin Si
  Mettre à jour les coûts provisoires des voisins du noeud v_j et *insérer* ces derniers dans la file de priorité appropriée selon la fonction de coût $\big(\delta(s, v_j) + \hat{\delta}(v_j, e)$ pour Q_F et $\hat{\delta}(s, v_j) + \delta(v_j, e)$ pour $Q_B\big)$.
Fin Tant que
Mettre à jour la liste des routes potentielles $Liste_{RP}$ en supprimant tous les itinéraires dans lesquels aucun arrêt n'a été marqué comme visité par les deux recherches.

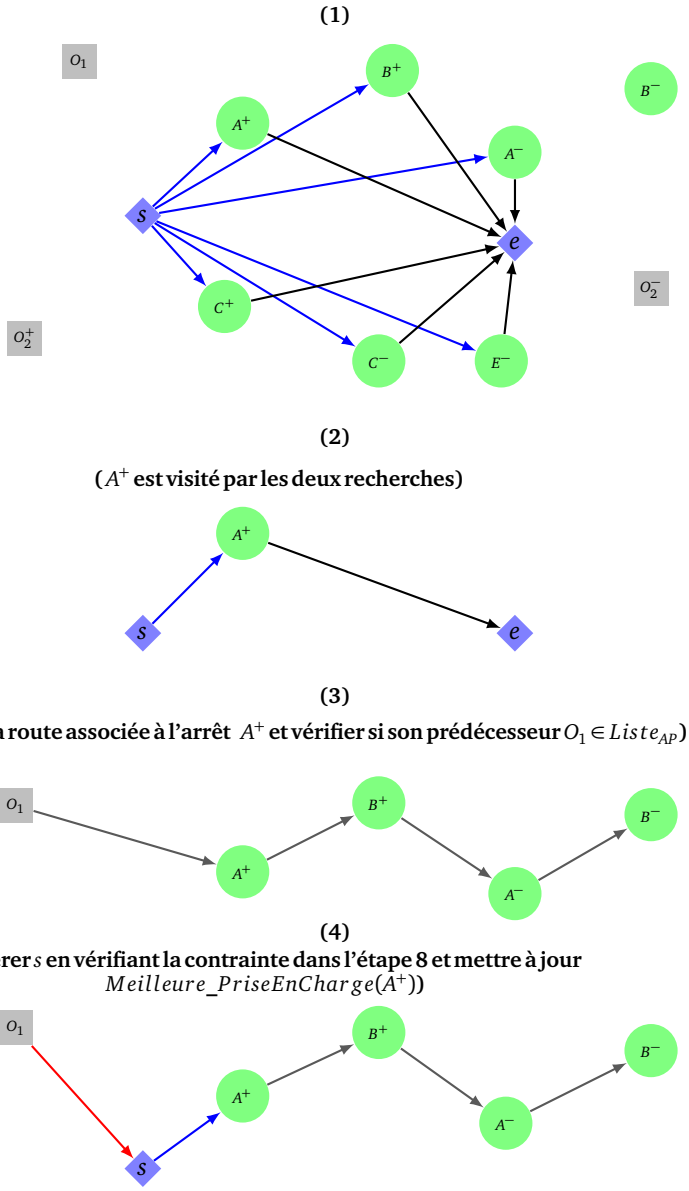


FIGURE 5.8 – La sous-figure (1) projette les plus courts chemins qui sont déterminés par l'algorithme BA (les arcs de couleur bleue correspondent aux plus courts chemins déterminés par la recherche avant et les arcs de couleur noir la recherche arrière). Tous les chemins déterminés ont une durée plus petite ou égale à $\lambda_r \delta(s,e)$. Une fois qu'un arrêt est visité par les deux recherches comme présenté dans la sous-figure (2), nous récupérons la route associée à ce conducteur. Ensuite, nous vérifions si depuis le prédécesseur de l'arrêt A^+ (c'est à dire O_1), ce conducteur est capable de prendre en charge le nouveau client à s (c'est à dire, vérifier la condition $O_1 \in Liste_{AP}$, voir sous-figure (3)). Enfin, l'heure d'arrivée maximale de chaque passager à bord est vérifiée dans le but de valider l'insertion de ce nouveau point de prise en charge. Pour l'arrêt A^+, nous gardons le coût d'insertion (c'est à dire $\delta(O_1,s)+\delta(s,A^+)-\delta(O_1,A^+)$) dans $Meilleure\_PriseEnCharge(A^+)$ (voir sous-figure (4)).

L'algorithme 21 permet de mettre à jour l'ensemble des étiquettes *Meilleure_PriseEnCharge*. Pour cela, nous parcourons chaque route potentielle $P = v_0, v_1, \ldots, v_{r-1}$ ($P \in Liste_{RP}$) par ordre croissant des arrêts à partir du successeur du point de prise en charge. Les noeuds n'appartenant pas à la liste *Liste_Potentielles_PD* ne disposeront pas d'étiquettes car l'insertion du nouveau point de dépose e après ces noeuds n'est pas admissible. La figure 5.9 montre un exemple de mise à jour effectuée sur une route potentielle.

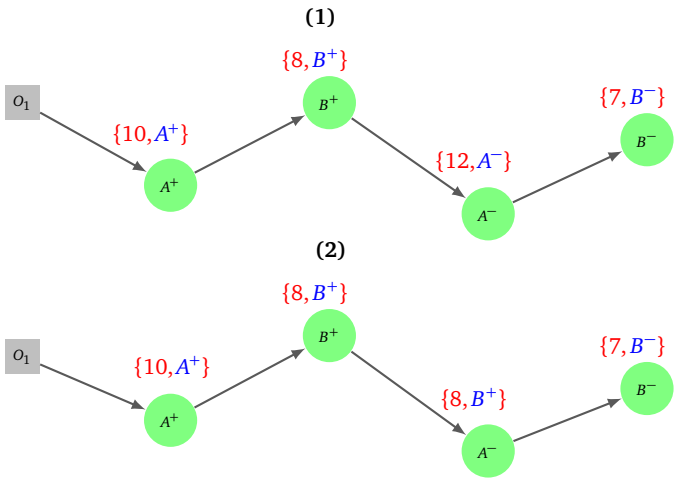


FIGURE 5.9 – Les deux valeurs sur chaque noeud correspondent respectivement au coût d'insertion (couleur rouge) et la position dans laquelle l'insertion est faite (couleur bleu). Dans la sous-figure (1), chaque couple au-dessus du noeud correspond à l'information sur l'insertion effectuée juste avant l'arrêt en question. Dans la sous-figure (2), nous prenons en considération toutes les insertions possibles qui peuvent se produire jusqu'à l'arrêt concerné en gardant celle avec le coût minimal. Par exemple, la meilleure insertion qui peut être effectuée avant l'arrêt A^- correspond à l'insertion entre A^+ et B^+ avec un coût égal à 8.

Durant l'opération de parcours, la durée $D_r^{dest->arrets}$ est mise à jour au fur et à mesure (voir la figure 5.10). Cette dernière correspond à la plus grande durée restante avant de dépasser l'heure maximale de dépose de chaque passager à bord du véhicule. Ceci permet de limiter l'espace de recherche effectuée dans l'étape suivante.

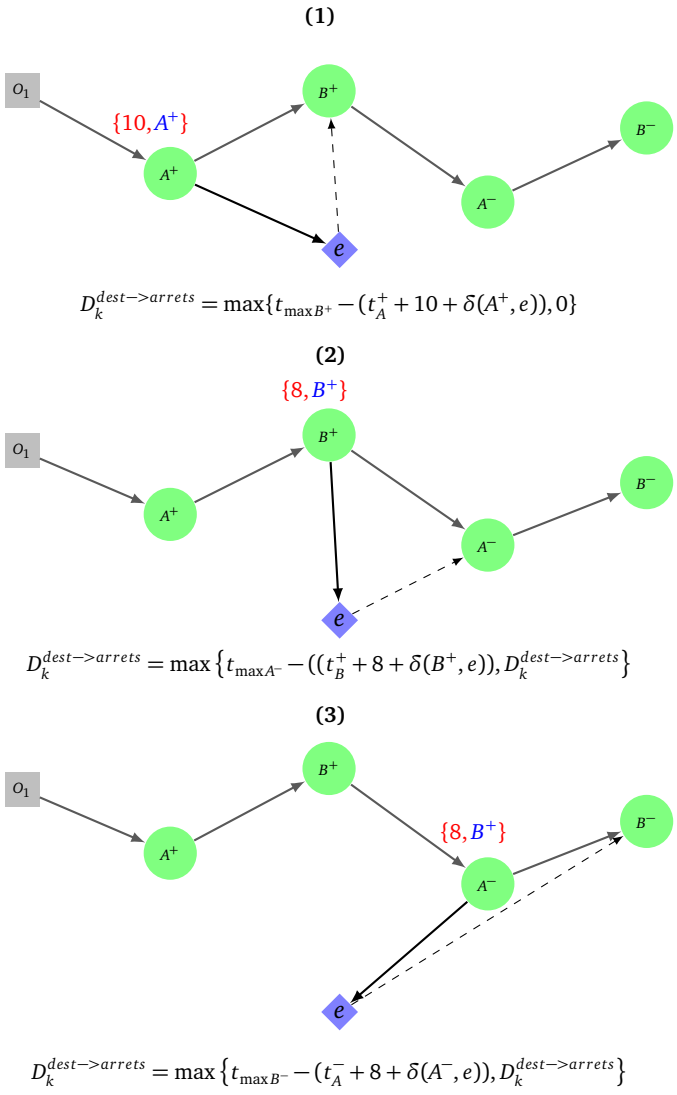


FIGURE 5.10 – Détermination de la durée $D_r^{dest->arrets}$ lors de la mise à jour des routes potentielles. De même, les noeuds n'appartenant pas à la liste *Liste_Potentielles_PD* ne seront pas pris en compte dans la détermination de la durée $D_r^{dest->arrets}$.

Pour un conducteur de taxi, une première tentative d'insertion du point de dépose e après avoir déposé tous les passagers à bord est effectuée (ligne 17 de l'algorithme 21).

Algorithme 21 Mise à jour de l'ensemble $Meilleure\_PriseEnCharge$

Entrée: Graphe G, lieu de prise en charge du client s, destination e, D_r^{dest},
$Liste_{RP}$: Liste des routes potentielles.
$Meilleure\_PriseEnCharge(j)$: Meilleure insertion du nouveau lieu de prise du client avant l'arrêt j, $\forall j \in Liste_{RP}$
$Charge\_max(v_j)$, $\forall v_j \in Liste_{RP}$
Sortie: Mise à jour des étiquettes $Meilleure\_PriseEnCharge(j)$ $\forall j \in Liste_{RP}$
$Pbest$: Meilleure insertion pour le lieu de prise en charge et de dépose du nouveau client dans un itinéraire d'un conducteur.
$D_r^{dest->arrets}$: La plus grande durée restante avant de dépasser le temps maximal d'arrivée de chaque passager à bord du véhicule concerné.

1: **Initialisation :** $Pbest \leftarrow null$, $D_r^{dest->arrets} \leftarrow 0$
2: **Pour tout** $P = v_0, v_1, \ldots, v_{r-1} \in Liste_{RP}$ **Faire**
3: Soit i le successeur du nœud de prise en charge dans la route P
4: **Pour** $j = i, \ldots, r-1$ **Faire**
5: **Si** $Charge\_max(v_j)$= Faux **Alors**
6: *Meilleure_PriseEnCharge(j)*=$\min\{Meilleure\_PriseEnCharge(j), Meilleure\_PriseEnCharge(j-1)\}$
7: {Nous gardons également la position de l'arrêt i qui minimise le coût d'insertion}
8: **Si** $\left(t_j + Meilleure\_PriseEnCharge(j) + \delta(v_j, e)\right) < t_{\max r}$ **Alors**
9: $D_r^{dest->arrets} \leftarrow \max\{t_{\max j+1} - \left(t_j + Meilleure\_PriseEnCharge(j) + \delta(v_j, e)\right), D_r^{dest->arrets}\}$
10: **Fin Si**
11: **Sinon**
12: $Meilleure\_PriseEnCharge(j) \leftarrow -1$
13: **Fin Si**
14: **Fin Pour**
15: **Si** v_{r-1} correspond à un arrêt d'un taxi **Alors**
16: Récupérer la meilleure insertion i du lieu de prise en charge s du nouveau client avant l'arrêt $r-1$ (c'est à dire l'étiquette $Meilleure\_PriseEnCharge(r-1)$)
17: **Si** $\delta(s, v_{i+1}) + \sum_{m=i+1}^{r-1} \delta(v_m, v_{m+1}) + \delta(v_{r-1}, e) \leq D_r^{dest}$ **Alors**
18: Mettre à jour $Pbest$ si nécessaire
19: **Fin Si**
20: **Fin Si**
21: **Fin Pour**

5.6.3 Insertion du lieu de dépose du nouveau client dans les routes potentielles

A ce niveau, toutes les insertions possibles du point de prise en charge du nouveau client dans les différentes routes potentielles sont testées. En se basant sur ces routes, nous pouvons déterminer la position optimale pour insérer la destination *e* du nouveau client de manière à réduire au maximum le coût de l'insertion.
Deux situations peuvent se produire après la prise en charge du nouveau client. La première situation correspond à celle où le client est directement servi après sa prise en charge. Dans la seconde situation, le conducteur n'a pas le temps d'amener directement le nouveau client à sa destination *e* car il doit d'abord servir d'autres passagers. Il est à rappeler que les passagers potentiels qui peuvent être pris en charge ou déposés juste après la dépose du nouveau client sont ceux qui appartiennent à la liste *Liste_Potentielles_PD*. Ainsi, pour vérifier les insertions possibles restantes, d'autres plus courts chemins doivent être calculés à partir du lieu de dépose *e* vers chaque arrêt v_j appartenant à la liste *Liste_Potentielles_PD*. Ces insertions sont déterminées en utilisant l'algorithme de Dijkstra borné, dans lequel nous limitons l'expansion jusqu'à ce que la durée maximale $D_r^{dest->arrets}$ soit atteinte.

L'algorithme 22 détaille les opérations effectuées dans cette étape.

Algorithme 22 Insertion du lieu de dépose e du nouveau client dans une route potentielle

Entrée: Graphe G, destination e, $D_r^{dest->arrets}$
$Liste_{AP}$: liste des positions et des arrêts potentiels des conducteurs
$Liste_{RP}$: liste des routes potentielles.
$RD_j = \min\{t_{\max i} - t_i, i = j, \ldots, r-1\}$ la plus petite durée restante avant de dépasser le temps maximal d'arrivée de chaque passager i à bord du véhicule. $0 \leq j \leq r-1$
$Meilleure\_PriseEnCharge(j)$: meilleure insertion du lieu de prise en charge du nouveau client après l'arrêt j, $\forall j \in Liste_{RP}$
Sortie: Pbest : meilleure insertion du lieu de prise en charge et de dépose du nouveau client dans la route d'un conducteur.

Initialisation : Q.*Insérer*(e), Recherche ← Faux
Tant que Recherche = Vrai **Faire**
  $v_j \leftarrow Q.\min()$
  Marquer v_j comme visité et retirer ce dernier dans la file de priorité Q
  Si $\delta(e, v_j) \leq D_r^{dest->arrets}$ **Alors**
    Si $v_j \in Liste_{RP}$ et $v_j \in$ *Liste_Potentielles_PD* **Alors**
      Récupérer la route appropriée $P = v_0, \ldots, v_j, \ldots, v_{r-1}$ de la liste $Liste_{RP}$ et son conducteur associé k
      Si $v_{j-1} \in$ *Liste_Arrêts_Potentiels* **and** $\big(\delta(v_{j-1}, e) + \delta(e, v_j) - \delta(v_{j-1}, v_j) +$ *Meilleure_PriseEnCharge(j-1)*$\big) \leq RD_j$ **Alors**
        Mettre à jour la meilleure insertion Pbest si nécessaire
      Fin Si
      {Tester l'insertion du lieu de dépose e directement après s c'est à dire v_{j-1}, s, e, v_j}
      Si $v_{j-1} \in Liste_{AP}$ **et** $\big(\delta(v_{j-1}, s) + \delta(s, e) + \delta(e, v_j) - \delta(v_{j-1}, v_j)\big) \leq RD_j$ **Alors**
        Mise à jour de la meilleure insertion $Pbest$ si nécessaire
      Fin Si
    Fin Si
    Mettre à jour les coûts provisoires des voisins du noeud v_j et *insérer* ces derniers dans la file de priorité Q.
  Sinon
    Recherche ← Faux
  Fin Si
  Mettre à jour la charge du véhicule l_i, RD_i et le temps d'arrivée t_i à chaque arrêt i associé à l'itinéraire Pbest
Fin Tant que

La figure 5.11 montre le fonctionnement de ce procédé lorsqu'un arrêt est marqué par la recherche.

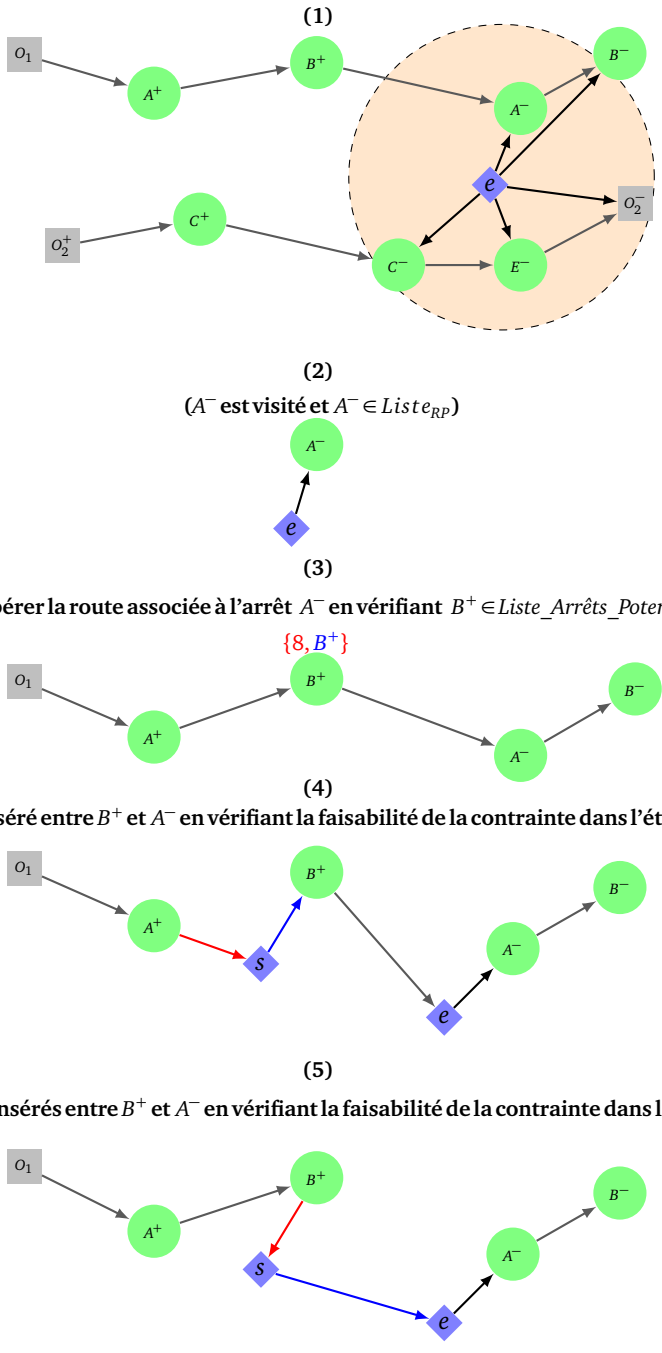


FIGURE 5.11 – La sous-figure (1) montre l'espace de recherche des plus courts chemins déterminés par l'algorithme de Dijkstra borné. Tous les chemins déterminés ont une durée plus petite que $D_r^{dest->arrets}$. Une fois qu'un arrêt est visité (sous-figure (2)), nous récupérons la route associée (sous-figure (3)) dans le but de tester deux insertions différentes. La première insertion concerne uniquement le lieu de dépose e. Ce dernier est inséré entre le noeud visité (A^-) et son prédécesseur (B^+), tandis que le lieu de prise en charge s est inséré dans sa position optimale ($Meilleure\_PriseEnCharge(B^+) = \{8, B^+\}$) (voir sous-figure(4)). Pour la deuxième insertion (sous-figure (5)), le lieu de prise en charge et de dépose du nouveau client sont insérés entre le nœud visité (A^-) et son prédécesseur (B^+). Pour cela, nous vérifions si depuis le prédécesseur de l'arrêt B^+ (c'est à dire A^-), (i) le conducteur est capable de prendre en charge le nouveau client à s (c'est à dire vérifier la condition $B^+ \in Liste_{AP}$) et (ii) la contrainte du temps d'arrivée maximal pour chaque passager à bord est respectée.

Cette approche est facilement extensible pour prendre en compte le coût du trajet de chaque passager. Pour cela, dans l'algorithme 21 nous vérifions que le partage des coûts entre les passagers à bord et le nouveau client avant la mise à jour des étiquettes *Meilleure_PriseEnCharge*(v_j) est possible à chaque arrêt v_j appartenant à la liste des routes potentielles $Liste_{RP}$.

5.6.4 Complexité de l'approche

L'étape de "Prise en charge" est assez similaire à celle décrite dans la première approche. Cependant, contrairement à celle-ci, l'insertion du lieu de prise en charge et de dépose se fait parallèlement avec la découverte des chemins. Plus exactement, l'insertion du lieu de prise en charge s'effectue en utilisant l'algorithme de Dijkstra bidirectionnel, suivi d'une mise à jour de l'ensemble des meilleurs arrêts de prise en charge *Meilleure_PriseEnCharge* sur l'ensemble des routes potentielles. L'insertion de la destination de la nouvelle requête est basée sur la procédure de marquage utilisée dans l'algorithme de Dijkstra. Enfin, la complexité de cette deuxième approche est réduite à trois algorithmes de Dijkstra avec quelques parcours de routes des taxis potentiels.

5.7 Comparaison des deux approches

Comme on peut le constater d'après les deux complexités temporelles associées aux deux approches, la deuxième approche est plus rapide que la première. En effet, dans la deuxième l'insertion se fait au même temps que la découverte des plus courts chemins. En revanche, la première approche est facilement adaptable à d'autres fonctions objectifs ce qui n'est pas le cas de la deuxième. Plus spécifiquement, cette deuxième approche est directement basée sur la minimisation de la durée d'insertion du lieu de prise en charge et de dépose. Ceci permet d'avoir une insertion en même temps que la détermination des plus courts chemins. Cette dernière nécessite également le stockage d'une information supplémentaire sur chaque arrêt de chaque conducteur en circulation. Il s'agit de la plus petite durée restante avant de dépasser le temps maximal d'arrivée de chaque passager à bord du véhicule (c'est à dire, l'ensemble *RD*)). D'où la nécessité de garder les deux approches, sans que l'une domine l'autre.

5.8 Expérimentations

5.8.1 Réseau routier

Le réseau routier utilisé dans ce chapitre correspond à celui de la ville de New-York City récupéré sur OpenStreetMap. Ce dernier est composé de 1 289 838 de noeuds et 5 153 919 d'arcs. (voir la figure 5.12).

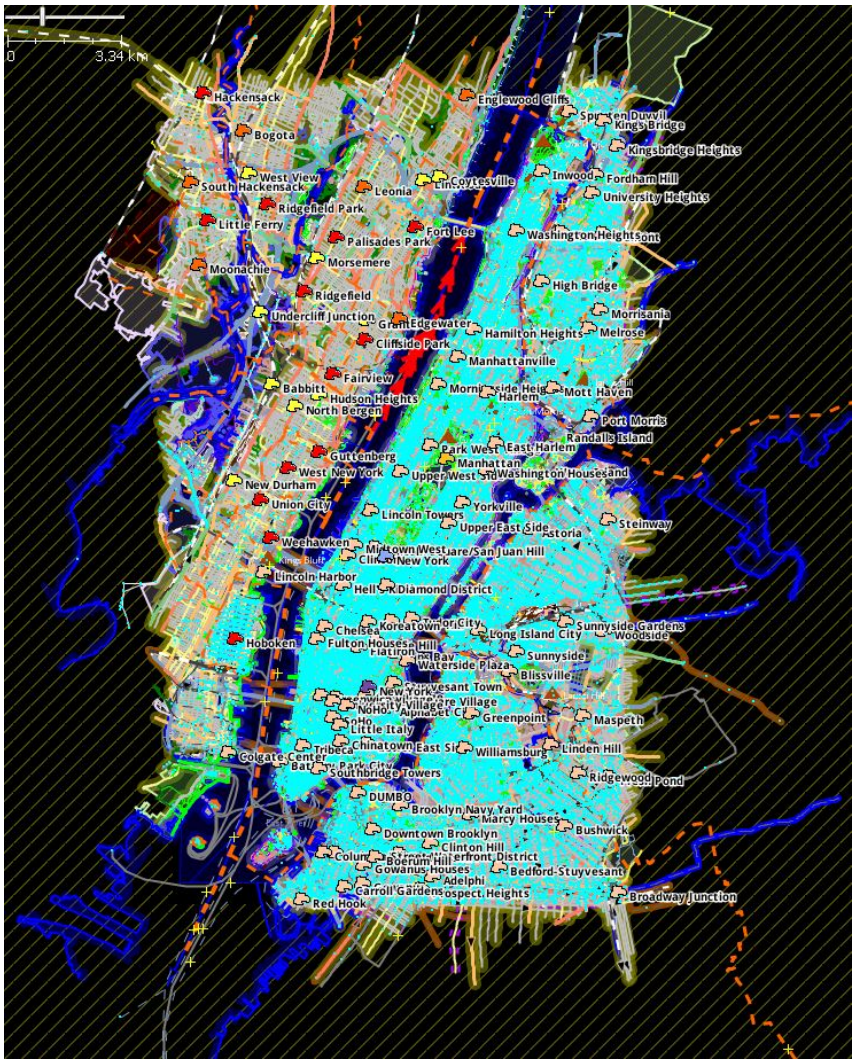


FIGURE 5.12 – Le réseau routier correspondant à la ville de New-York City.

Un point important à préciser sur le calcul d'itinéraire $(v_1, v_2, \ldots, v_n)$ renvoyé par "Osmsharp" [22] concerne la possibilité d'avoir une plus courte durée en sommant les durées des sous-trajets deux à deux du trajet en question. C'est à dire, $\delta(v_1, v_n) \geq \sum_{i=1}^{n} \delta(v_i, v_i + 1)$. En effet, lorsque "Osmsharp" calcule l'ensemble des plus courts chemins deux à deux, il ne prend pas en compte les durées effectuées sur les différents virages et ronds points lorsqu'il passe de l'un à l'autre. En revanche, lorsque ce dernier calcule un plus court chemin entre les deux extrémités du trajet, les durées effectuées sur

22. http ://www.osmsharp.com/

les différents ronds-points et virages sont prises en compte. Ainsi, une nouvelle requête est insérée dans un itinéraire d'un taxi ayant déjà des passagers à bord est dans laquelle cette nouvelle requête se trouve sur l'un des plus courts chemins des sous-trajets du taxi. Ces passagers peuvent alors avoir une nouvelle durée qui est plus courte que celle avant l'insertion du nouveau client. Dans nos calculs on prend on compte ces lacunes de sorte que les nouvelles durées des passagers à bord du taxi n'aient pas une durée plus petite afin d'éviter ces biais dans nos résultats.

5.8.2 Données des taxis

Nos données ont été extraites à partir d'une base de données contenant les trajectoires des taxis cabines à New York. Cette dernière couvre quatre années d'exploitation de taxis à New York (2010-2011-2012-2013) et elle est composée de 697 622 444 trajets.
La figure 5.13 représente un sous-ensemble de données de trajets de taxis. Chaque ligne correspond à un trajet d'un taxi occupé. Il est à noter que ces données brutes contiennent un grand nombre d'erreurs. Par exemple, il y a plusieurs trajets où les distances rapportées sont significativement plus courtes que les distances à vol d'oiseau, ce qui contredit un des postulats de la géométrie euclidienne. Dans certains cas, la durée "trip_time_in_secs" est mesurée en secondes, alors que dans d'autres cas, cette dernière est représentée en minutes. Pour éviter ceci, le temps de trajet peut être calculé correctement lorsque l'on soustrait "pickup_datetime" à "dropoff_datetime". En outre, de nombreux trajets comportent des coordonnées GPS de (latitude, longitude)=(0, 0).

medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag	pickup_datetime	dropoff_datetime	passenger_count	trip_time_in_secs	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
2010000001	2010000001	VTS	1		2010-01-01 00:00:00	2010-01-01 00:34:00	1	34	14.05	-73.948418	40.72459	-73.92614	40.864761
2010000002	2010000002	VTS	1		2010-01-01 00:00:00	2010-01-01 00:33:00	1	33	9.65	-73.997414	40.736156	-73.997833	40.736168
2010000003	2010000003	VTS	1		2010-01-01 00:00:00	2010-01-01 00:07:00	1	7	1.63	-73.967171	40.764236	-73.956299	40.781261
2010000004	2010000004	VTS	1		2010-01-01 00:00:00	2010-01-01 00:33:00	1	33	26.61	-73.789757	40.646526	-74.136749	40.601543
2010000005	2010000005	VTS	1		2010-01-01 00:00:00	2010-01-01 00:28:00	2	28	3.15	-73.99955	40.731152	-73.977448	40.763031
2010000006	2010000006	VTS	1		2010-01-01 00:00:00	2010-01-01 00:27:00	1	27	11.15	-73.993698	40.736946	-73.861435	40.756256
2010000007	2010000007	VTS	1		2010-01-01 00:00:00	2010-01-01 00:18:00	3	18	4.30	-74.006058	40.739925	-73.957405	40.765686
2010000008	2010000008	VTS	1		2010-01-01 00:00:00	2010-01-01 00:27:00	1	27	9.83	-73.874245	40.773739	-74.0028	40.760498
2010000009	2010000009	CMT	1	0	2010-01-01 00:00:00	2010-01-01 00:18:13	1	18.219999999999999	3.40	-74.004868	40.751656	-73.988342	40.718399
2010000010	2010000010	CMT	1	0	2010-01-01 00:00:02	2010-01-01 00:36:27	2	36.420000000000002	12.40	-73.95546	40.787731	-73.961739	40.666935

FIGURE 5.13 – Les premières colonnes d'un sous-ensemble de données de taxis.

D'après l'étude [Donovan and Work, 2015], ces erreurs représentent environ 7,5% de tous les trajets. Actuellement, seules les données brutes (non corrigées) sont disponibles sur leur site [23].

23. http ://hafen.github.io/taxi/

5.8.3 Élaboration du scénario et des objectifs de l'étude

Il est à rappeler que chaque ligne du tableau correspond à une requête satisfaite par un taxi. L'idée est donc de parcourir les requêtes selon l'ordre croissant de l'heure de prise en charge dans le but d'insérer les requêtes dans les itinéraires des taxis déjà en circulation. La première requête (c'est à dire, la ligne correspondant à la plus petite heure de prise en charge) sera considérée comme étant un trajet de taxi avec le nombre de passagers associé. La deuxième requête (succédant la première requête) sera insérée dans le premier taxi déjà en circulation en utilisant notre algorithme d'insertion. Dans le cas où l'insertion de cette nouvelle requête n'est pas admissible, un nouveau taxi (celui qui correspond à la ligne concernée) sera ajouté dans le réseau afin de satisfaire cette requête. Le processus est donc répété pour chaque ligne. La situation de rejet peut se produire lorsqu'une nouvelle requête n'a pas pu être insérée dans l'un des taxis en circulation et que le taxi associé à cette nouvelle requête n'est plus disponible (ou bien déjà affecté à une autre requête précédente). Ainsi, pour une requête qui n'a pas pu être insérée dans un des taxis déjà en circulation, elle ne sera pas automatiquement considérée comme étant un nouveau trajet de taxi. Pour ce faire, on doit d'abord vérifier si le taxi associé à cette nouvelle requête est disponible, sinon on doit la rejeter.
Ce scénario nous permet de mesurer les performances suivantes :

(i) Les coûts économisés en partageant un même taxi comparés aux coûts des taxis-cabines.
(ii) Le nombre de kilomètres économisés en utilisant le système de taxis-partagés.
(iii) Le nombre de taxis réduit sur le réseau.
(iv) Le temps d'insertion d'une nouvelle requête dans une des trajectoires des taxis.

L'obtention du point (ii) nécessite la reconstruction des trajectoires des taxis. Comme mentionné ci-dessus, chaque ligne correspond à un trajet d'un taxi occupé. Cependant, lorsque le taxi est inoccupé, sa trajectoire n'est pas connue. Une des solutions pour y remédier est d'estimer une trajectoire plausible entre son lieu de dépose e_i et le lieu de la prochaine prise en charge s_{i+1}. Pour ce faire, on calcule pour chaque arrêt de dépose, la différence de durée entre l'instant de la prochaine prise en charge et l'instant de la dépose actuelle. Si cette durée correspond à la durée du trajet direct depuis le lieu de dépose vers le lieu de la nouvelle prise en charge, alors le temps d'attente sur le lieu de prise en charge est nul. Autrement, le temps d'attente du taxi sur le lieu de prise en charge correspondra à la durée restante avant l'heure de la nouvelle prise en charge. De notre processus de reconstruction de trajectoire de taxis résulte une borne inférieure sur sa vraie trajectoire.

Exemple du déroulement du processus dynamique d'insertion

Dans cette partie, nous présentons notre processus dynamique d'insertion par un exemple illustratif. Afin de simplifier cette présentation, nous avons récupéré 4 requêtes de trajets à partir de la base de données des taxis NYC, notées respectivement par r_1, r_2, r_3 et r_4. La figure 5.14 représente les 4 requêtes ainsi que les informations correspondantes.

Comme décrit précédemment, la première requête r_1 sera automatiquement considérée comme un taxi avec le passager A à bord. Autrement dit, le taxi transporte le passager A depuis son lieu de prise en charge A^+ vers son lieu de dépose A^-. Les requêtes restantes vont être considérées comme des nouveaux clients à insérer dans le trajet r_1. La première étape tente d'insérer le client de la requête r_2 dans la trajectoire du taxi r_1. Pour cela, nous devons faire avancer le taxi r_1 jusqu'à l'heure d'entrée de la requête r_2 dans le système, c'est à dire à l'instant t_2 : $6h04$. Une fois que la deuxième requête est insérée avec succès, nous passons à la requête suivante avec la plus petite heure de prise en charge. Cette dernière correspond à la requête r_3. Le même raisonnement se poursuit en tentant d'insérer cette dernière dans la trajectoire du taxi r_1 ayant déjà à bord les deux passagers A et B. Dans notre cas, cette requête ne peut être insérée dû au long détour effectué par les deux passagers A et B. Ainsi,

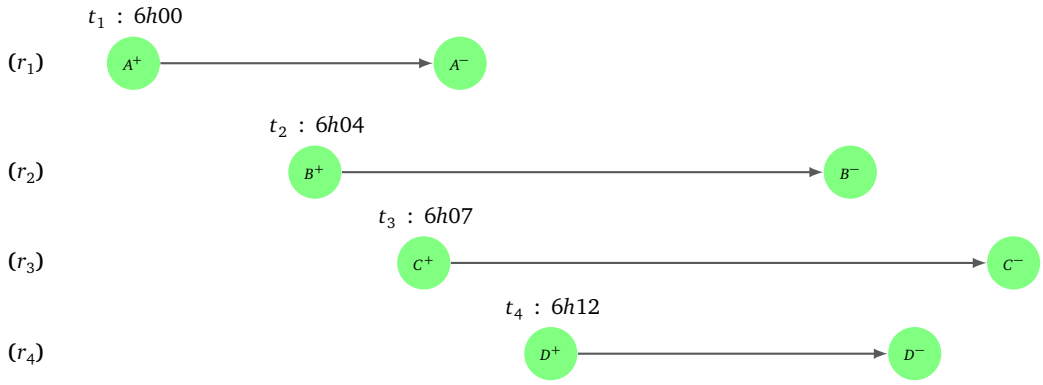


FIGURE 5.14 – Les 4 requêtes (r_1, r_2, r_3 et r_4) récupérées à partir de la base de données de taxis New-yorkais. Sur chaque trajectoire d'une requête, on retrouve l'heure de départ associée.

nous ne considérons pas automatiquement cette requête comme un nouveau taxi avec le passager C à bord. Nous devons d'abord vérifier si son taxi n'a pas été affecté ailleurs. Dans cet exemple, nous devons juste vérifier si le taxi affecté apriori à la requête r_3 n'est pas le même que celui de la requête r_1. Si cela est vérifié, le passager C sera pris en charge par son propre taxi, sinon la requête r_3 sera rejetée. A ce niveau, on aura donc deux taxis en circulation (voir la figure 5.15).

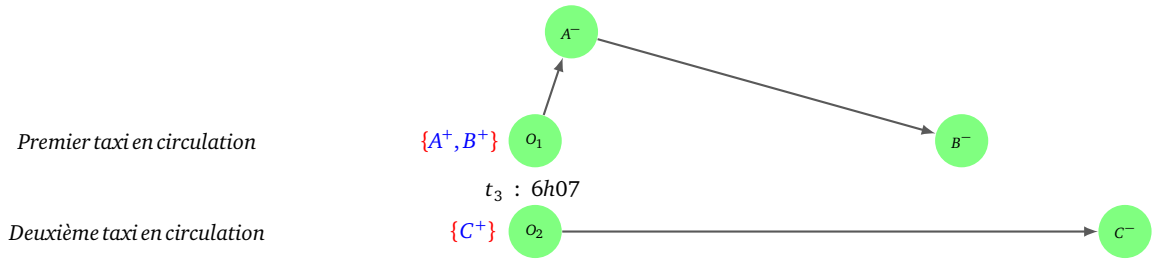
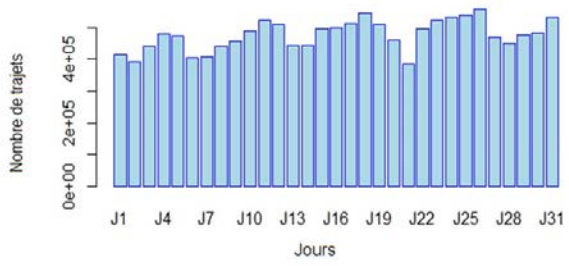


FIGURE 5.15 – Les deux taxis en circulation à l'instant t_3, le premier ayant deux passagers à bord (passagers A et B). En revanche, le deuxième taxi transporte le passager C. Les noeuds O_1 et O_2 représentent la postition du premier et du deuxième taxi à l'instant t_3.

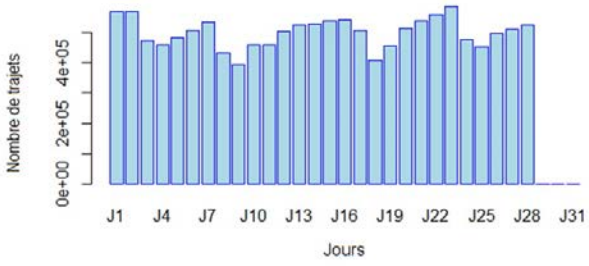
Le même raisonnement s'applique pour la requête r_4 en tentant dans un premier temps de l'insérer dans l'une des routes des deux taxis avant de lui attribuer son propre taxi si cela est possible.

5.8.4 Analyse de données des taxis NYC et la détermination d'une plage horaire de test

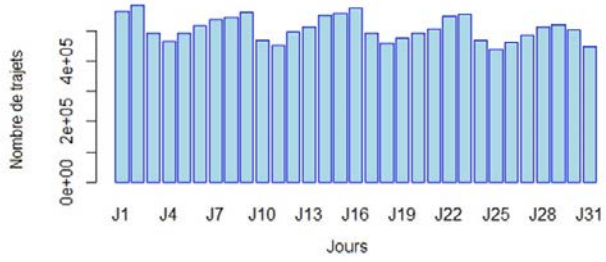
Vu la dimension de la base de données, une analyse est nécessaire sur cette dernière afin de choisir une plage horaire adéquate sur les quatre années 2010-2013. Notre test s'est porté sur une plage horaire précise de l'année 2013 contenant un nombre de trajets suffisant et bien réparti entre eux.
Les figures ci-dessous présentent le nombre de trajets de taxis effectués par jour sur l'ensemble des mois de l'année 2013.



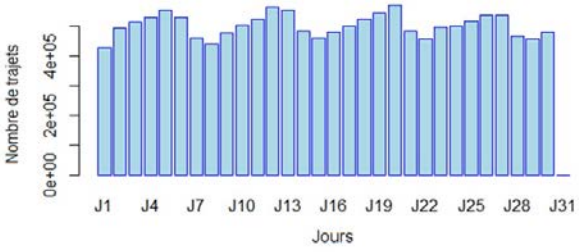
(a) Mois de Janvier



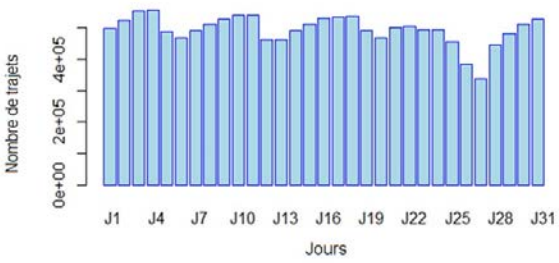
(b) Mois de Février



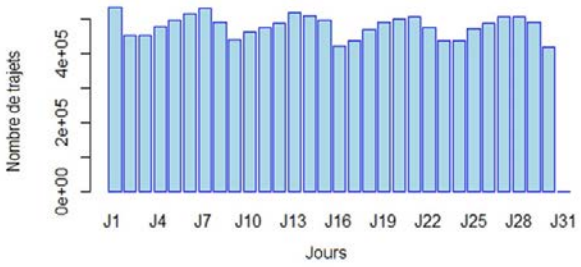
(c) Mois de Mars



(d) Mois d'Avril

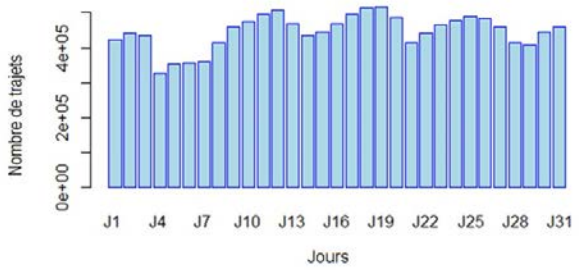


(e) Mois de Mai

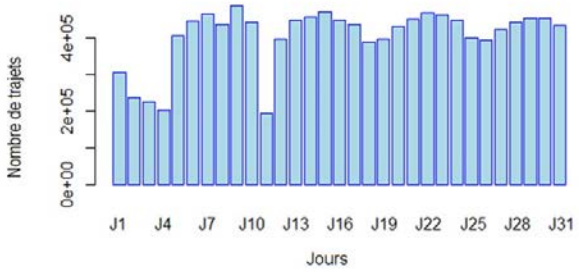


(f) Mois de Juin

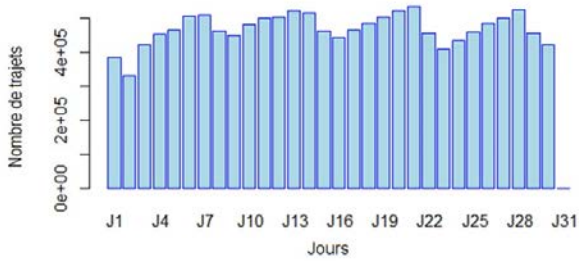
FIGURE 5.16 – Le nombre de trajets pour les premiers mois de l'année 2013 (Janvier, Février, Mars, Avril, Mai et Juin).



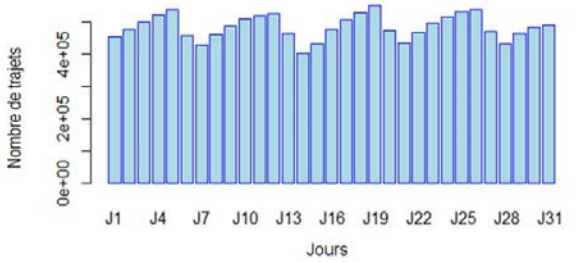
(a) Mois de Juillet



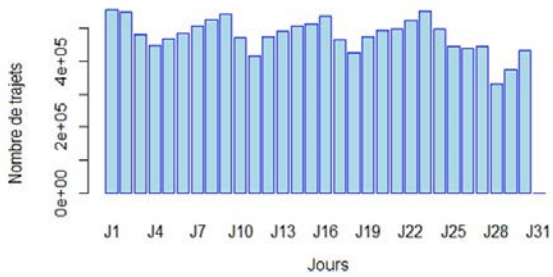
(b) Mois d'Août



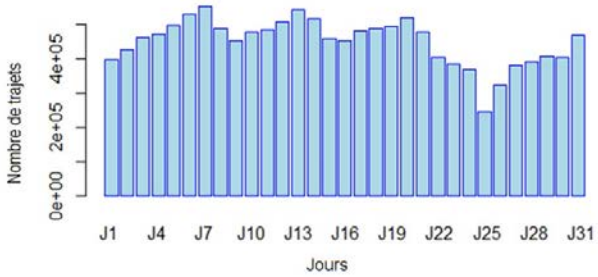
(c) Mois de Septembre



(d) Mois d'Octobre



(e) Mois de Novembre



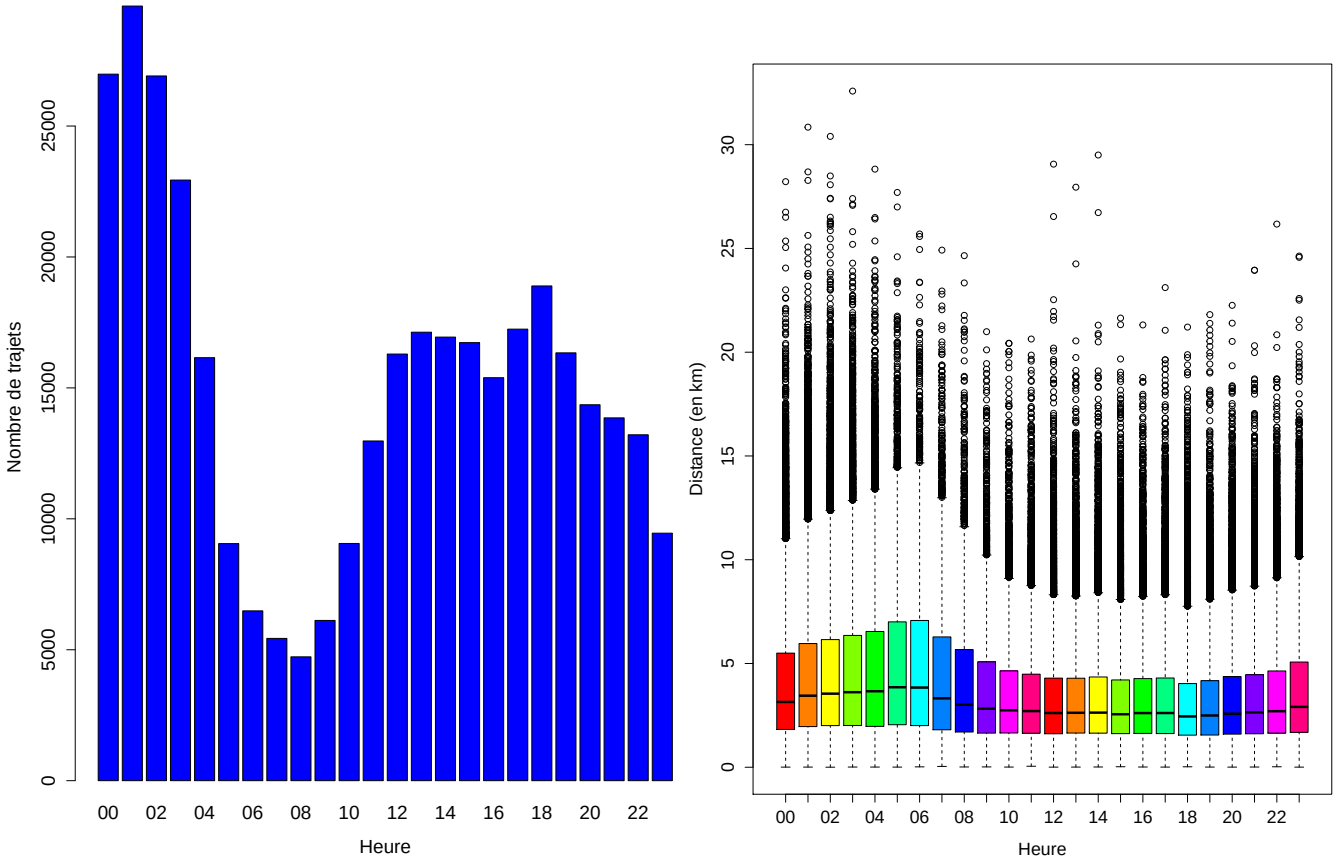
(f) Mois de Décembre

FIGURE 5.17 – Le nombre de trajets pour les six derniers mois de l'année 2013 (Juillet, Août, Septembre, Octobre, Novembre et Décembre).

Comme on peut le constater d'après les figures 5.16 et 5.17, le nombre de trajets par jour sur l'ensemble des mois est très important. Notre choix s'est porté sur le 01 Janvier 2013.

Analyse par heure de la journée du 1 Janvier 2013

Afin d'avoir plus de chance de former des équipages composés de plus d'un passager, nous avons analysé les distances des trajets parcourues sur les différentes heures.

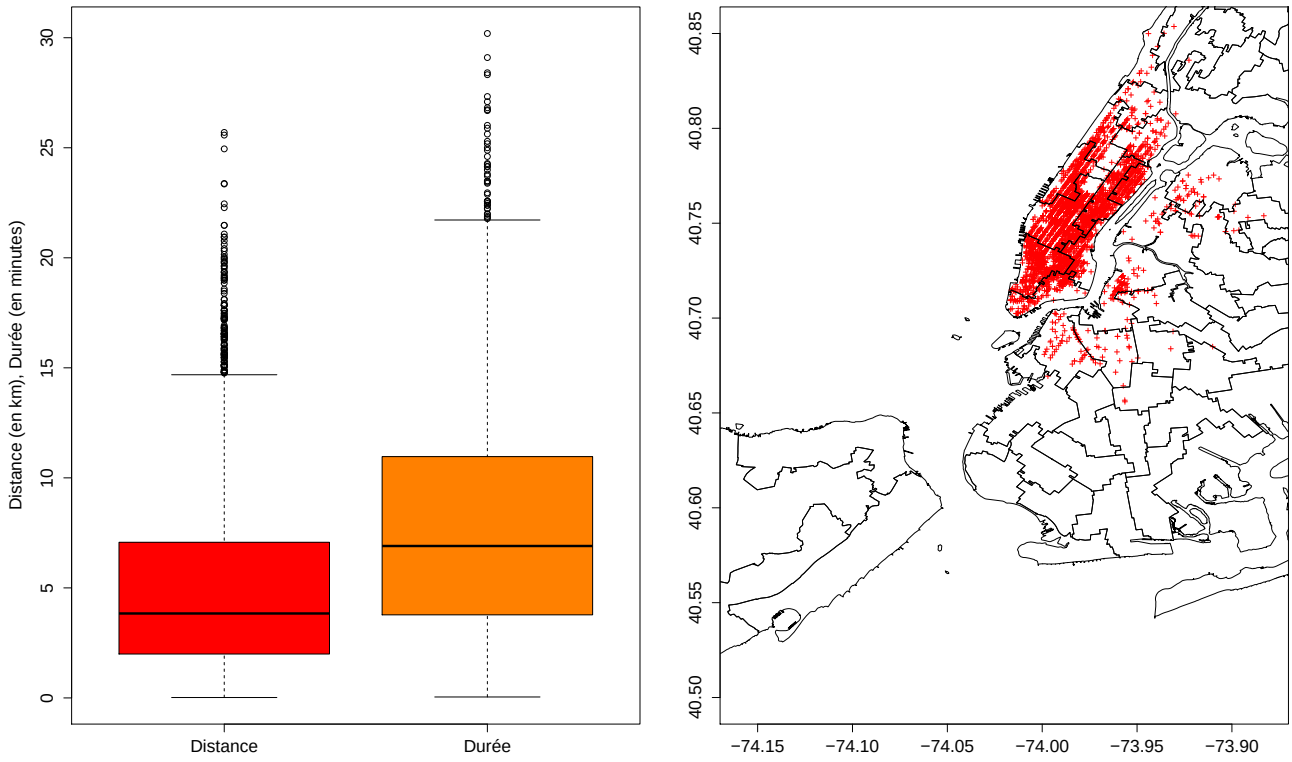


(a) Nombre de trajets par heure

(b) Boîtes à moustaches sur les distances pour chaque heure

FIGURE 5.18 – Le nombre de trajets et les boîtes à moustaches sur les distances pour chaque heure de la journée du 1 Janvier 2013.

Les médiane des distances des requêtes sont proches les unes des autres. Ainsi, notre plage horaire est fixée entre 6h00 et 07h00. Sur cette plage horaire, la circulation n'est pas dense ce qui implique que les durées calculées par notre système sont équivalentes aux durées réelles. En les isolant, on se retrouve à 6489 trajets.
Dans la figure 5.19 nous présentons les boîtes à moustaches sur les durées des trajets ainsi que la répartition spatiale des requête se trouvant dans la plage horaire [6h,7h].



(a) Boîtes à moustache sur les distances/durées des trajets.

(b) Les requêtes des passagers.

FIGURE 5.19 – Boîtes à moustache sur les distances/durées des trajets ainsi que les requêtes des passagers dans la plage horaire [06h,07h]

Le tableau 5.1 présente quelques statistiques sur les données d'entrées récupérées sur cette plage horaire en supprimant les données aberrantes qui n'ont pas pu être projetées sur notre réseau routier. Comme informations obtenues, on retrouve le nombre de requêtes satisfaites, le nombre de taxis utilisés, la distance totale parcourue et la durée totale sur l'ensemble des trajets des taxis. Il est à noter que la procédure utilisée pour la reconstruction de la trajectoire d'un taxi est décrite dans la section 5.8.3.

Distance totale parcourue en kilomètres	Nombre de taxis utilisés	Durée totale en heures	Nombre de requêtes satisfaites
38748.18	3124	1031.531	6489

TABLE 5.1 – Quelques statistiques sur les données brutes d'entrées.

Le nombre maximal de clients transportés par un taxi sur la plage horaire fixée [06h, 07h] ne dépasse pas 7. La valeur médiane sur le nombre de clients transporté par un taxi est de 2 clients.

5.8.5 Résultats numériques

Dans cette partie, nous exposons nos résultats numériques mesurés sur la plage horaire [6h, 7h]. Le temps de détour maximal de chaque passager est fixé à 30% de la durée de son trajet direct et chaque requête (ligne du tableau) comporte un seul client. Le temps d'attente maximal $maxwait$ des requêtes est fixé à 10 minutes.
Dans un premier temps, nous présentons les performances suivantes :

(i) Le nombre de taxis utilisés en appliquant le système de taxis partagés.
(ii) La distance totale parcourue par les taxis en appliquant le système de taxis partagés.
(iii) Le nombre et le pourcentage de requêtes rejetées.

Ces performances sont mesurées dans deux cas différents : lorsque le coût de référence de chaque client est mis à jour à chaque nouvelle insertion et lorsque le coût de référence est fixé au coût du trajet s'il voyageait seul.

Coût de référence variable

Le tableau 5.2 présente quelques performances de notre système lorsque le coût de référence est mis à jour à chaque nouvelle insertion.

Taxis utilisés		Distance totale parcourue		Durée totale		Requêtes rejetées	
Nombre	Pourcentage (%)	Nombre (km)	Pourcentage (%)	Nombre (heures)	Pourcentage (%)	Nombre	Pourcentage (%)
1713	54.83	29583.4	76.34	737.5	71.49	605	9.32

TABLE 5.2 – Performances obtenues en appliquant le système de taxis partagés.

En offrant aux taxis la possibilité de transporter plusieurs passagers en même temps, le nombre de taxis sur le trafic est considérablement réduit. En effet, ce nombre est réduit à 1713 taxis au lieu de 3124 sans le concept de partage de taxis. Ce qui représente 45.17% de réduction sur la flotte totale. De même, la distance totale parcourue est de 29583.4 km et la durée totale est de 737.5 heures. Ces réductions par rapport au système classique de taxis sont très significatives et atteignent 23.66% pour la distance totale et 28.51% pour la durée totale. Si le coût de trajet est fixé à 1 euro le kilomètre, ce dernier atteindra 2,9 milles euros d'économies générées en l'espace d'une heure sans compter les coûts fixes des chauffeurs de taxis.
Comme précisé précédemment, notre scénario engendre le rejet de requêtes. Le nombre de requêtes rejetées est de 605 sur 6408, ce qui ne dépasse pas 10%. Ce nombre reste inférieur au nombre de taxis réduit par le système de taxis-partagés. Ainsi, dans le pire des cas les requêtes rejetées peuvent être considérées comme satisfaites par ces taxis.
Le tableau 5.3 présente en détail les partitions de requêtes satisfaites.

# requêtes insérées à la fin des taxis	# requêtes insérées en partageant le trajet	# requêtes satisfaites avec leur propre taxi
2378	1793	1713

TABLE 5.3 – Partitions des requêtes satisfaites.

Malgré les courtes distances des trajets, le nombre de requêtes insérées en partageant leur trajet reste significatif. Dans le cas contraire, notre système privilégie l'insertion de nouveaux clients dans les taxis déjà en circulation avant de leur attribuer leur propre taxi s'il n'est pas affecté ailleurs. Dans le tableau 5.4, nous présentons la distance parcourue par les différents équipages formés.

Composition des groupes	Distance totale parcourue (en km)	(en %)
1 passager	20160	68.14
2 passagers	3258	11
3 passagers	437	1.5
4 passagers	54	0.2

TABLE 5.4 – Distance parcourue par les différents groupes.

La colonne "Distance totale parcourue" représente la distance totale parcourue en kilomètres ainsi que le pourcentage comparé aux données brutes d'entrées pour les différents groupes. Le nombre de trajets composés de 4 passagers est faible par rapport aux autres groupes dû aux petites distances des trajets (dont la médiane correspondante est de 3.83 km).
La distance restante 5674 km (∼19%) est parcourue par les taxis lorsqu'ils sont inoccupés. Plus spécifiquement, elle représente le cumul des distances parcourues par les taxis entre une dernière dépose et une nouvelle prise en charge, ce qui revient en moyenne à 3.3 km par heure pour chaque taxi. Afin d'avoir une idée sur le pourcentage des coûts économisés par requête, nous présentons dans la figure 5.20 une boîte à moustache regroupant les différents gains dans chaque quartile.

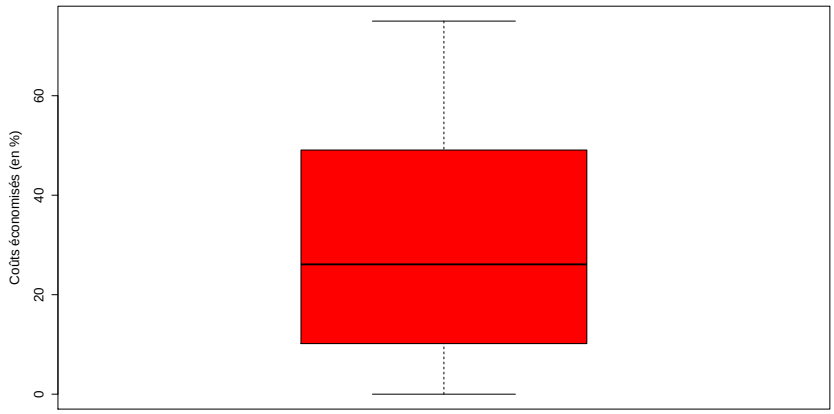


FIGURE 5.20 – Les différents coûts économisés (en %) pour les requêtes insérées avec succès.

Les économies générées sont significatives avec une médiane de 26% et une moyenne de 28%. On atteint également des économies à la hauteur de 75% du trajet direct. Ce dernier correspond généralement aux groupes composés de 4 personnes.

Coût de référence fixe

Dans cette partie nous relaxons la contrainte sur le coût de référence de chaque client. Autrement dit, le coût de référence de chaque client correspond à son coût s'il voyageait seul et ce dernier reste

fixe pour chaque nouvelle insertion. Ceci nous permet de savoir si la mise à jour du coût de référence à chaque nouvelle insertion engendre beaucoup de rejets sur le partage de taxis. Le tableau 5.5 projette quelques sorties lorsque le coût de référence est fixe pour chaque client.

Taxis utilisés		Distance totale parcourue		Durée totale		Requêtes rejetées	
Nombre	Pourcentage (%)	Nombre (km)	Pourcentage (%)	Nombre (heures)	Pourcentage (%)	Nombre	Pourcentage (%)
1675	53.61	29365.47	75.78	724	70.19	576	8.87

TABLE 5.5 – Performances obtenues en appliquant le système de taxis partagés avec un coût de référence fixe.

En fixant le coût de référence de chaque client, quelques améliorations en termes de nombre de taxis utilisés, de distance totale parcourue, durée totale et nombre de requêtes rejetées ont été observées. Plus exactement, 38 taxis supplémentaires ont été réduits sur le trafic, le nombre de requête rejetées a diminué de 30 et la distance totale parcourue de 217.93 km.
Dans le tableau 5.6 nous présentons la nouvelle répartition des requêtes satisfaites.

# requêtes insérées à la fin des taxis	# requêtes insérées en partageant le trajet	# requêtes satisfaites avec leur propre taxi
2230	2008	1675

TABLE 5.6 – Partitions des requêtes satisfaites.

On voit clairement une augmentation du nombre de requêtes insérées en partageant leur trajet (215 requêtes supplémentaires).
Le tableau 5.7 présente la distance parcourue pour chaque groupe.

Composition des groupes	Distance totale parcourue (en km)	(en %)
1 passager	19851.88	67.60
2 passagers	3406.07	11.59
3 passagers	570.50	1.94
4 passagers	70.01	0.23

TABLE 5.7 – Distance parcourue par les différents groupes.

On observe une légère augmentation de 148 km pour le groupe de 2 passagers, 133 km pour le groupe de 3 passagers et 16 km pour le groupe de 4 passagers. La distance parcourue par un taxi lorsqu'il est inoccupé est en moyenne de 3.2 km par heure.

Dans la figure 5.21 nous présentons les différents pourcentages des coûts économisés.

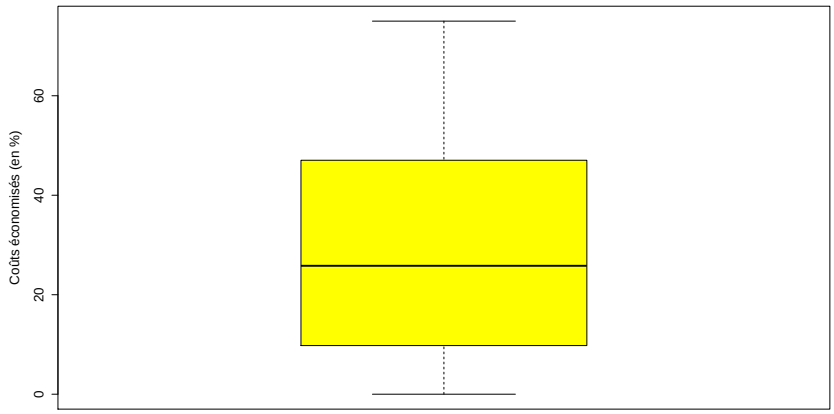


FIGURE 5.21 – Les différents coûts économisés (en %) pour les requêtes insérées avec succès en fixant les coûts de référence.

Ces derniers sont légèrement plus petits que lorsqu'on ne fixe pas les coûts de références. La médiane est de 25.85% et la moyenne correspondante de 26.18%.
En fixant les coûts de référence, les gains économisés devraient être moins importants que lorsque les coûts de références sont variables. En effet, dans le premier cas les gains peuvent décroître lors de nouvelles insertions, alors que dans le second cas les gains ne feront qu'augmenter. En revanche, ceci est vrai uniquement lorsque le taxi peut transporter au maximum deux passagers. En mettant à jour le coût de référence, on risque de refuser un client car le coût économisé décroît. Alors que lorsque l'on accepte ce dernier (malgré la réduction du coût économisé), on peut avoir un troisième client qui augmentera les coûts économisés (grâce au deuxième client accepté) et vice-versa. Afin de remédier à ceci, une approche stochastique peut sembler intéressante pour prédire d'éventuelles nouvelles requêtes avant d'avoir une meilleure décision sur le rejet ou l'acceptation des clients ce qui engendre une réduction au niveau des coûts économisés. Cette étude ne rentre pas dans le cadre de notre projet.

Dans ce qui suit, toutes les performances présentées correspondent au cas où le coût de référence de chaque client est mis à jour à chaque nouvelle insertion.

Temps d'attente des taxis avant une nouvelle prise en charge

Dans notre scénario, une fois qu'un taxi dépose son dernier client, il reste sur place en attendant d'éventuelles nouvelles requêtes. Afin de mesurer la sensibilité de cette décision, nous avons mesuré le nombre de taxis ayant attendu :

- 0 minute avant une nouvelle prise en charge.
- Moins de 30 secondes avant une nouvelle prise en charge.
- Entre 30 secondes et 1 minute avant une nouvelle prise en charge.
- Plus d'une minute avant une nouvelle prise en charge.

0 minute	Moins de 30 secondes	Entre 30 secondes et 1 minute	> 1 minute
351	1355	4	3

TABLE 5.8 – Nombre de taxis immobilisés en attendant une nouvelle prise en charge.

Comme on peut le constater d'après le tableau 5.8, la plupart des taxis attendent moins de 30 secondes avant une nouvelle prise en charge. Cela est dû au nombre élevé de demandes de trajets. L'attente supérieure à 1 minute des taxis avant une nouvelle prise en charge n'est pas forcément dû à la non arrivée de nouvelles requêtes mais également aux présences d'autres taxis aux alentours. En effet, sur ces trois taxis (c'est à dire, ceux qui n'ont pas eu une nouvelle prise en charge en moins d'une minute), deux d'entre eux ont eu au moins une nouvelle requête sur un rayon de 5 minutes. Ainsi, malgré le nombre élevé de requêtes, le besoin de répartition des taxis sur un périmètre donné est nécessaire pour une meilleure gestion des flux.
La figure 5.22 projette les arrivées des nouvelles requêtes avec un laps de temps de 5 minutes dans le but de montrer la faible fluctuation entre ces dernières.

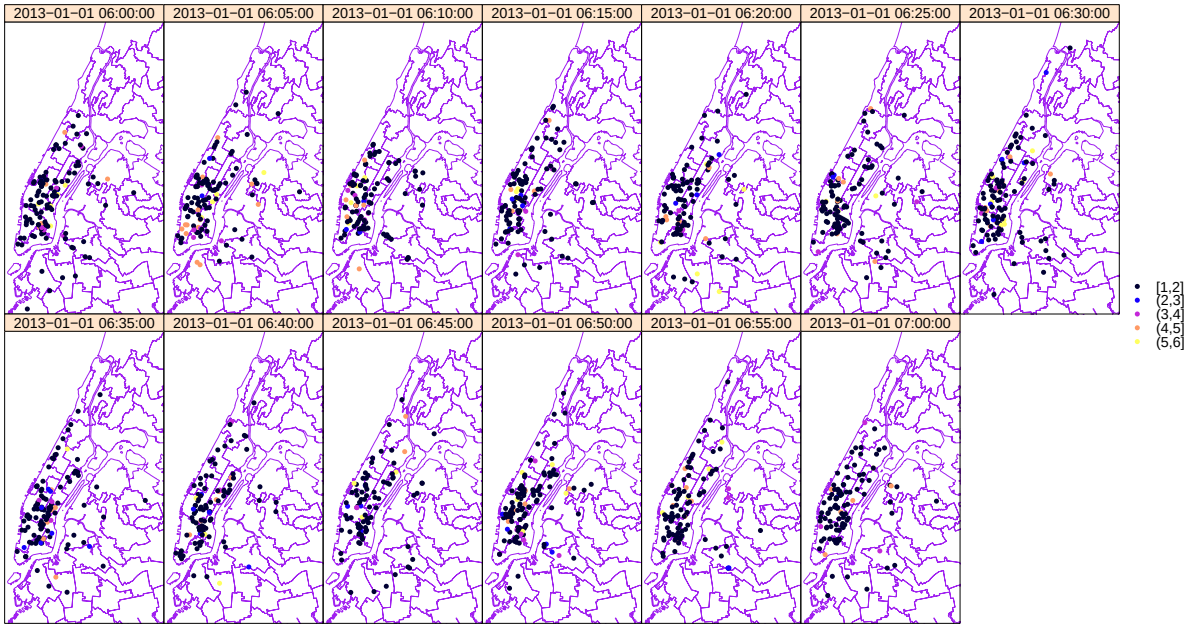


FIGURE 5.22 – Les nouvelles requêtes avec un laps de temps de 5 minutes. Chaque point sur la carte représente un nombre spécifique de passagers représentés par une couleur.

La figure 5.23 présente le nombre de nouvelles requêtes arrivées à chaque minute dans la fenêtre de temps [6h, 7h]. Comme on peut le constater, la fluctuation est faible et le nombre de requêtes est assez proche à chaque minute. Le gap ne dépasse pas 47 requêtes.

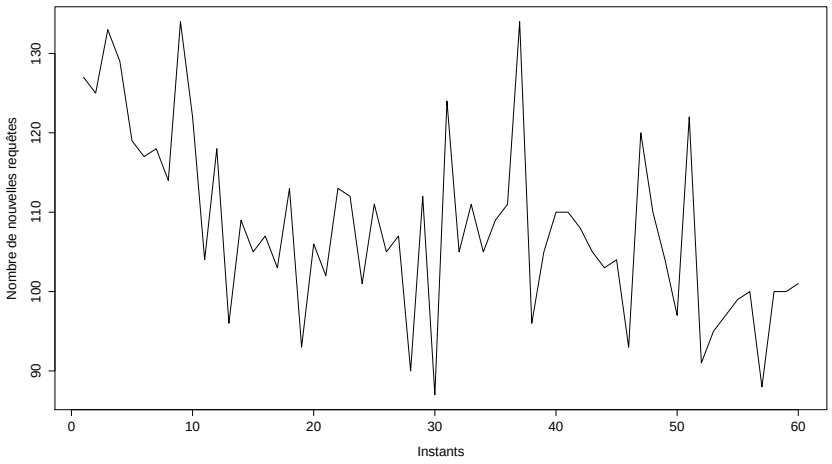


FIGURE 5.23 – Le nombre de nouvelles requêtes à chaque minute dans la fenêtre horaire [6h, 7h].

Sur les 1713 taxis utilisés, nous avons mesuré le nombre de kilomètres parcourus et le nombre de passagers additionnels servis entre [6h, 7h]. 1263 des taxis utilisés ont augmenté leur nombre de passagers servis d'une moyenne de 4. En revanche, malgré l'augmentation des requêtes servies, pour 661 d'entre eux leur distance parcourue a baissé en moyenne de 8 km. Cela est dû en partie à la non circulation aléatoire des taxis à la recherche de nouvelles requêtes lorsqu'ils sont inoccupés. Le nombre de passagers additionnels par taxi atteint 10 passagers et le nombre moyen de kilomètres supplementaire est de 11 km. Sur les 450 taxis restants, leur nombre de passagers diminue en moyenne de 1 et la réduction en termes de distance parcourue est de 7 km.

Temps de traitement des requêtes

A ce niveau, nous présentons le temps de traitement d'une requête en fonction du nombre de taxis potentiels ainsi que leur nombre d'arrêts.
La figure ci-dessous présente le temps d'exécution d'une requête pour la première approche en fonction du nombre de routes potentielles et du nombre d'arrêts des conducteurs. Comme on peut le constater, le temps de traitement d'une requête augmente quasi-linéairement avec le nombre de routes potentielles ainsi que le nombre d'arrêts de prise en charge et de dépose. Le temps de traitement maximal d'une requête n'excède pas 21 secondes. La médiane correspondante est de 9.37 secondes.

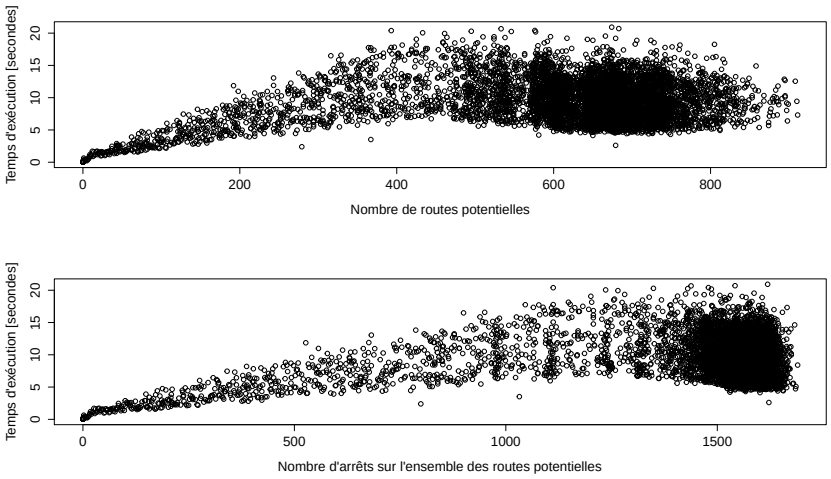


FIGURE 5.24 – Temps d'exécution en fonction du nombre de routes potentielles et du nombre d'arrêts.

Dans la figure 5.25, nous présentons le temps de traitement écoulé dans la procédure de calcul des 4 plus courts chemin et dans la procédure d'insertion du lieu de prise en charge et de dépose.

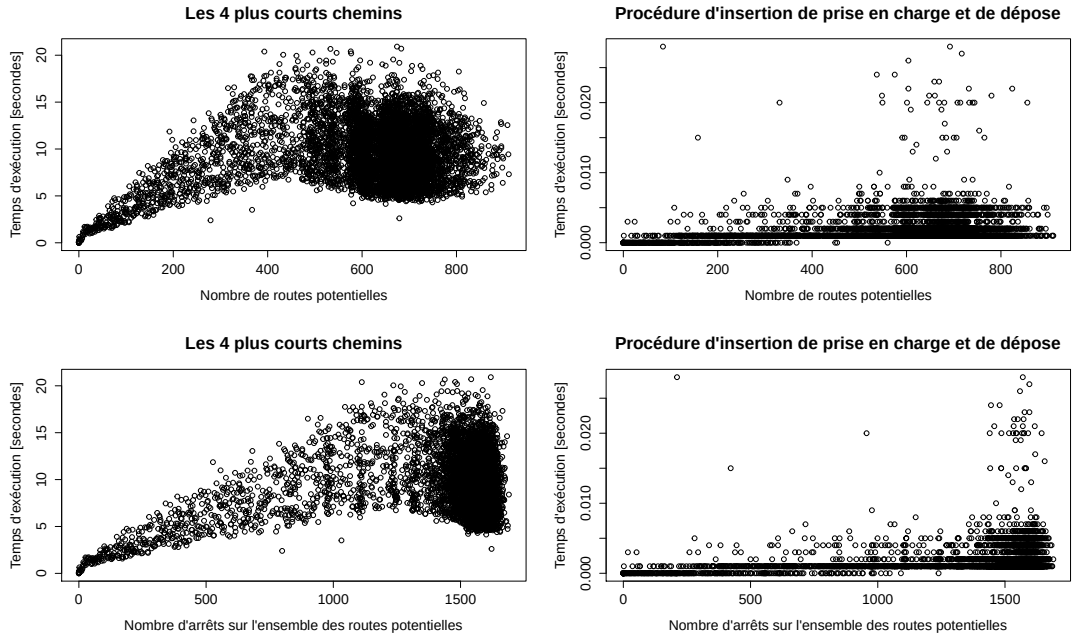


FIGURE 5.25 – Temps d'exécution pour la procédure de calcul des 4 plus courts chemins ainsi que la procédure d'insertion en fonction du nombre de routes potentielles et d'arrêts des conducteurs.

D'après la figure 5.25, la phase de calcul des 4 plus courts chemines (en incluant la détermination des bornes inférieures) représente 98% du temps d'exécution d'une requête.
Nous comparons également les deux approches en termes de temps de traitement de requêtes. La figure 5.26 présente le temps de traitement d'une requête en utilisant la deuxième approche. Dans cette dernière, le procédure d'insertion du nouveau lieu de prise en charge et de dépose s'effectue parallèlement avec la découverte de chemins.

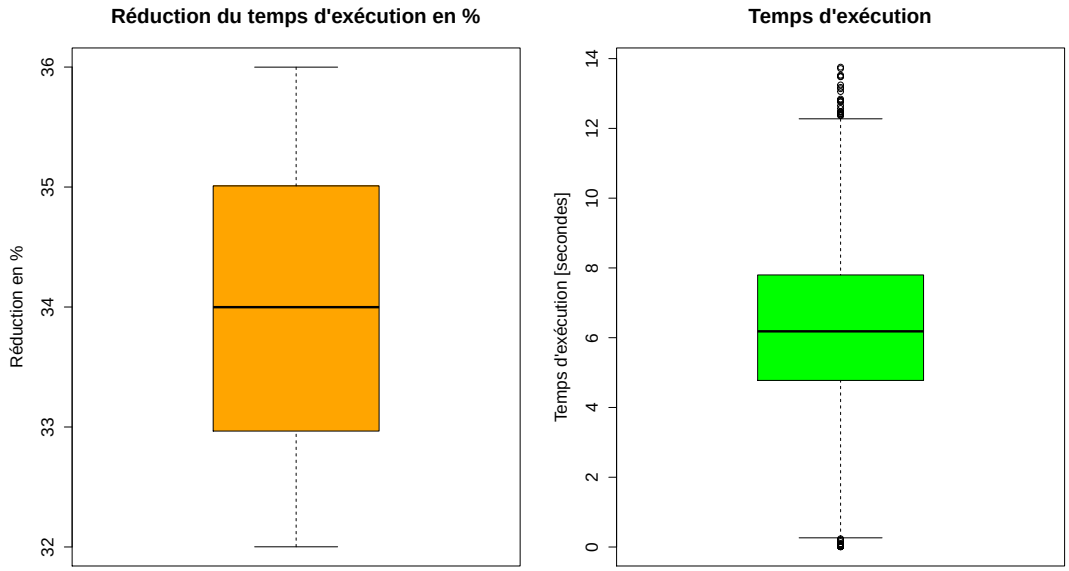


FIGURE 5.26 – Temps d'exécution pour la deuxième approche.

En utilisant la deuxième approche, la réduction du temps de traitement d'une requête atteint au maximum 36%. Ainsi, le temps de traitement d'une requête n'excède pas 15 secondes (contre 21 secondes pour la première approche). Le temps de traitement d'une requête a légèrement baissé en moyenne de 3.66 secondes par requête. Ceci est également dû à une meilleure détermination des bornes inférieures qui limitent l'expansion de calculs des trois plus courts chemins (contrairement à 4 plus courts chemins pour la première approche).

5.9 Conclusion du chapitre

Dans ce chapitre nous avons proposé deux approches différentes pour la résolution du problème dynamique de Dial-A-Ride en temps réel et dans lequel aucun pré-traitement sur le calcul des durées et des distances n'est nécessaire. Des contraintes réelles ont été prises en compte, à savoir : le coût et la durée de trajet de chaque client. Ces approches ont été validées dans un cadre réaliste (sur un vrai réseau routier et des données réelles) sans aucune perturbation au niveau des données d'entrée. Comme on a pu le constater dans nos expérimentations, le concept de partage de taxis est bénéfique pour : (i) les clients (car le coût de leur trajet est réduit en partageant un même taxi), (ii) pour les compagnies de taxis car les déplacements de leurs taxis ainsi que leurs flottes sont optimisés et (iii) pour l'environnement car cela réduit considérablement les effets de serre.
La décision sur la fixation ou non du coût de référence joue un rôle important sur le nombre de requêtes rejetées. Dans nos approches, nous avons opté pour une mise à jour de ces derniers pour attirer l'attractivité des clients. Autrement dit, les clients à bord d'un taxi donné n'accepterons pas l'ajout d'un *i-ème* client si leur coût économisé avant ce dernier est plus attractif.
Concernant les deux approches proposées, chacune d'entre elles tire un avantage particulier. La première approche est facilement adaptable pour d'autres fonctions objectifs, alors que la deuxième approche est basée sur une fonction objectif bien spécifique, d'où une plus grande efficacité sur le temps de traitement des requêtes.

6

Transport multimodal incluant le covoiturage dynamique

Sommaire

Ce chapitre décrit un problème de routage multimodal innovant, qui se produit à chaque fois qu'un utilisateur souhaite se déplacer d'un point A à un point B en utilisant le covoiturage, les transports publics ou la combinaison des deux. L'idée est de commencer par un itinéraire multimodal classique, puis de remplacer séquentiellement une ou plusieurs parties de cet itinéraire par un ou plusieurs trajets en covoiturage. Plus précisément, de la position de départ de la requête jusqu'à sa destination, on obtient le premier chemin multimodal classique. Parmi les différents arrêts composant ce trajet, nous ne considérons que les arrêts de correspondance où le passager doit faire un changement entre deux modes. Ces arrêts de correspondance sont considérés comme des lieux

potentiels pour le covoiturage. Nous définissons ainsi une estimation de proximité qui permet de comparer les itinéraires des conducteurs disponibles et l'itinéraire multimodal classique. Une fois que les conducteurs potentiels sont déterminés, on procède au processus de substitution. Ce dernier permet d'améliorer l'heure d'arrivée du passager à sa destination sous quelques contraintes, à savoir : le temps d'attente maximal pour le conducteur au lieu de prise en charge, ainsi que le temps maximal de détour du conducteur. Nous avons testé nos différentes approches sur un vrai réseau routier avec des données fournies par l'entreprise de covoiturage Covivo [24].

6.1 Introduction

La mobilité urbaine peut utiliser différents modes de transport : les transports publics ou les transports privés. Cette mobilité dépend à la fois des préférences et des disponibilités des utilisateurs. Nous proposons une approche algorithmique pour le problème multimodal en temps réel de type *"Earliest-Arrival Problem (EAP)"* dans un réseau de transport urbain en utilisant les transports publics et le covoiturage de sorte que la durée du trajet soit minimisée. Pour ce faire, nous commençons tout d'abord par déterminer un itinéraire multimodal classique, puis compte tenu des possibilités en covoiturage, nous essayons séquentiellement de substituer un sous-trajet en transport public à un trajet en covoiturage. Dans cet arrangement, les passagers soumettent des requêtes pour se déplacer d'un endroit à un autre, tandis que les conducteurs offrent des trajets de covoiturage. L'objectif d'un passager est de trouver un itinéraire plus rapide dans un contexte dynamique qui combine efficacement les différents modes de transport.
La seule étude qui considère le problème multimodal en incluant le covoiturage à été présentée dans [Bit-Monnot et al., 2013] sous le nom "The 2 synchronization points shortest path problem (2SPSPP)". Leur approche consiste à synchroniser deux itinéraires (celui du conducteur et du passager) en deux points du réseau de manière à ce que la durée cumulée des trajets soit minimisée. Ces deux points de synchronisation correspondent ainsi aux lieux de prise en charge et de dépose du passager. Leur approche est basée sur le calcul de plusieurs plus courts chemins dans un réseau multimodal.
Plus spécifiquement, le graphe parcouru par le conducteur est limité à celui associé à l'utilisation d'un véhicule personnel alors que le graphe parcouru par le passager est limité à celui modélisant les transports en commun et la marche à pied. Cette restriction se base d'une part sur une utilisation d'une expression régulière pour contraindre les séquences de modes valides, et d'autre part sur une modélisation du réseau de transport multimodal par un graphe orienté. L'algorithme permettant d'exclure certains modes de transport ou de limiter le nombre de transfert dans un réseau multimodal est présenté dans [Barrett et al., 2000].
Ce dernier n'est autre qu'une généralisation de l'algorithme de Dijkstra qui permet de déterminer des chemins valides en respectant les séquences fixées au préalable par l'utilisateur.
L'approche décrite dans [Bit-Monnot et al., 2013] détermine d'une façon optimale le lieu de prise en charge et de dépose entre un conducteur et un passager.
Plusieurs restrictions sont cependant présentes, à savoir :

- L'utilisateur doit choisir au préalable le conducteur avant l'exécution du processus.
  En effet, leur approche consiste seulement à déterminer les deux meilleurs emplacements intermédiaires pour un conducteur et un passager déjà fixés.
- Aucune contrainte de détour sur le conducteur n'est prise en compte. En effet, la minimisation de la durée cumulée du trajet ne garantit pas la minimisation du détour effectué par le conducteur.

24. www.covivo.fr

– La complexité quadratique de l'approche empêche son utilisation en contexte temps-réel dans des réseaux de grande taille.

Dans cette étude, nous considérons un cadre plus proche de la réalité en prenant en compte l'ensemble des conducteurs se trouvant dans le système à l'instant de l'arrivée d'une requête ainsi que la contrainte de détour des conducteurs. Pour une application en temps réel, nous réduisons également la complexité de l'approche en considérant les lieux de correspondance constituant le chemin multimodal classique comme lieux potentiels de prise en charge et de dépose du passager.
Cela nous permet de réduire l'espace de recherche ainsi que l'effort effectué par le passager en lui évitant des détours.
Nous nous positionnons dans la situation suivante : un utilisateur r appelé passager, souhaite se rendre d'une origine O_r vers une destination D_r à l'instant t_r. Son objectif est d'atteindre sa destination le plus rapidement possible, en utilisant les transports en commun, la marche à pied, le covoiturage, ou la combinaison des trois. Les horaires des transports en commun sont facilement accessibles par des API (Application Programming Interface). En Suisse, on peut utiliser Swiss public transport [25]. En France, un service similaire est disponible [26]. Le couple origine-destination (OD) des conducteurs potentiels est également scanné en temps réel lorsqu'une requête arrive dans le système.
Dans ce qui suit, nous allons présenter trois approches différentes pour l'insertion du covoiturage dans un itinéraire multimodal classique. Ces dernières seront respectivement notées : *approche directe*, *approche par étiquetage* et *approche par buckets*. Chacune des ces approches présente des avantages particuliers. Le déroulement de ce chapitre est comme suit : nous commençons par une description du problème ainsi que notre processus de substitution (section 6.2). La première approche (*approche directe*) est décrite dans la section 6.3. Les deux autres approches (*approche par étiquetage* et *approche par buckets*) sont respectivement présentées dans les sections 6.4 et 6.5. Ensuite, une évaluation comparative entre les trois approches ainsi que le système multimodal classique est présentée dans la section 6.6. Enfin, nos conclusions sont présentées dans la section 6.7.

6.2 Description et notations

Dans les différentes approches, nous travaillons seulement sur un réseau routier. Les informations concernant les transports publics sont récupérées à partir d'une API lorsqu'une requête entre dans le système. Plus précisément, le chemin multimodal classique récupéré par l'API est projeté sur le réseau routier. De cette façon, le graphe représentant le réseau de transport en commun n'est plus nécessaire. Le tableau 6.1 représente les notations utilisées dans ce chapitre.

25. http://transport.opendata.ch
26. http://www.navitia.io

TABLE 6.1 – Liste des notations

Notation	Définition
$s \rightsquigarrow e$	plus court chemin entre s et e.
$\delta(s,e)$	durée du plus court chemin de s à e.
$\hat{d}(s,e)$	distance à vol d'oiseau entre s et e.
$\hat{\delta}(s,e)$	durée estimée entre s et e t.q : $\hat{\delta}(s,e) = \frac{\hat{d}(s,e)}{v_{\max}}$, où $v_{\max}$ la vitesse maximale autorisée dans la zone concernée.
λ_k	coefficient de détour du conducteur k, $\lambda_k \geq 1$.
$\tau(x)_{ab}$	temps nécessaire pour un changement de modalité de a à b au niveau de l'arrêt x.
$t_a^r(x,m)$	l'heure d'arrivée à x en utilisant m comme mode de transport pour l'utilisateur r.
$t_d^r(x,m)$	l'heure de départ de x en utilisant m comme mode de transport pour l'utilisateur r.
$w_{\max}^k$	durée maximale d'attente fixée par le conducteur k à l'arrêt de prise en charge.

Dans ce qui suit, on notera par p une des modalités de transport multimodal classique, et par c la modalité en covoiturage.

6.2.1 Processus de substitution

Nous considérons seulement les chemins de substitution possibles qui n'augmentent pas le temps d'arrivée du passager à sa destination. Pour ce faire, nous introduisons la notion de *chemin de substitution admissible* avec la définition suivante :

Definition. 11 (Chemin de substitution admissible)
Un chemin (x,y) forme une substitution admissible pour le conducteur k et le passager r si et seulement si les trois contraintes suivantes sont respectées, à savoir :

$$t_a^r(x,p) + \tau(x)_{pc} - t_a^k(x,c) \leq w_{\max}^k \qquad \textit{temps d'attente} \tag{6.1}$$

$$\begin{aligned} \delta(O_k,x) + \max\left\{t_a^r(x,p) + \tau(x)_{pc} - t_a^k(x,c), 0\right\} \\ +\delta(x,y) + \delta(y,D_k) \leq \lambda_k \delta(O_k,D_k) \end{aligned} \qquad \textit{temps de détour} \tag{6.2}$$

$$\max\left\{t_a^k(x,c), t_a^r(x,p) + \tau(x)_{pc}\right\} + \delta(x,y) \leq t_a^r(y,p) \qquad \textit{temps d'arrivée} \tag{6.3}$$

La première contrainte concerne le temps d'attente du conducteur au lieu de prise en charge x. Ce temps doit être inférieur ou égal à la durée fixée par le conducteur. La seconde contrainte correspond au temps de détour effectué par le conducteur pour une prise en charge du passager r depuis x jusqu'à y en prenant en compte le temps d'attente effectué au lieu de prise en charge x. Ce dernier doit également respecter le détour maximal fixé par le conducteur. En revanche, la dernière contrainte assure au passager une arrivée à l'arrêt y plus rapide que celle utilisant les transports publics. Les contraintes (6.1), (6.2) et (6.3) prennent en compte le temps nécessaire pour passer du transport public au covoiturage sur le lieu de prise en charge.
La fonction objectif qui définit le meilleur chemin de substitution est basée sur le temps économisé. Plus spécifiquement, parmi tous les *chemins de substitution admissibles*, nous sélectionnons le *meilleur chemin de substitution* (x,y). Ce dernier génère un gain en temps d'arrivée plus élevé au passager s'il partage son trajet avec le conducteur k depuis x jusqu'à l'arrêt y plutôt que les transports publics.

6.3 Première approche : Approche directe

Nous présentons un exemple qui illustre les différentes étapes de notre approche. Cet exemple représente la situation suivante : un utilisateur r souhaite se déplacer d'une origine O_r vers une destination D_r à l'instant $t_r = 9:00$. Les transports publics lui permettent d'aller d'un point x_2 à un autre point x_{nbs}, respectivement à proximité de son point de départ O_r et de sa destination D_r. On notera x_2, x_3, ..., x_{nbs-1} les points de correspondance constituant son chemin multimodal classique. Pour simplifier l'exemple, on considère que l'origine du passager O_r correspond à un arrêt de bus, donc $O_r = x_1$. On considère également un seul conducteur (voir les figures : 6.4 à 6.9).

6.3.1 Étape d'initialisation

Lorsqu'une requête O_rD_r arrive dans le système à l'instant t_r, un plus court chemin multimodal classique ainsi que les chemins possibles en covoiturage entre les différents arrêts de correspondance sont déterminés. L'algorithme 23 détaille cette étape.

Algorithme 23 Traitement initial de la requête

Entrée: Demande O_rD_r, l'instant de départ t_r
Sortie: Chemin multimodal classique P,
Ensemble de plus courts chemins $PDrive$

1: Calculer un plus court chemin multimodal classique $P = O_r, x_1, \ldots, x_{nbs}, D_r$ en utilisant l'API, partant de O_r à l'instant t_r, avec une heure d'arrivée $t_a^r(x,p)$ et de départ $t_d^r(x,p)$, $x \in P$.
2: Calculer les plus courts chemins de substitution entre les différents arrêts P on associant le temps d'arrivée à chaque arrêt $t_a^r(x,c)$, $x \in P$.
3: **Initialisation** : $PDrive \leftarrow \emptyset$
4: **Pour tout** $x, y \in P$ tel que x est situé avant y **Faire**
5: **Si** $t_a^r(x,p) + \tau(x)_{pc} + \delta(x,y) \leq t_a^r(y,p)$ **Alors**
6: $PDrive \leftarrow PDrive \cup \{(x,y)\}$.
7: **Fin Si**
8: **Fin Pour**

En utilisant L'API, l'étape 1 détermine un chemin multimodal classique P avec les temps de chaque arrivée et de départ sur chacun des nœuds se situant entre la position de départ O_r et la destination D_r. L'étape 2 calcule les chemins possibles de substitution en voiture sur le long des arrêts du chemin P. Seuls ceux dont le temps d'arrivée en voiture à l'arrêt concerné est plus tôt que le temps d'arrivée en utilisant les transports publics sont conservés. Plus spécifiquement, l'heure d'arrivée au lieu de prise en charge plus le temps de changement nécessaire en incluant la durée du trajet jusqu'à l'arrêt de dépose y est comparé avec l'heure d'arrivée à y en transport public. Si le gain en temps pour l'utilisateur n'est pas positif, alors le trajet en covoiturage (x,y) ne sera pas conservé. Les trajets potentiels en covoiturage sont stockés dans l'ensemble $PDrive$.
La figure 6.1 illustre un exemple où un passager r se déplace à l'instant t_r=9h00 à partir de son point de départ O_r (qui est considéré comme le premier arrêt) vers sa destination D_r. Plus spécifiquement, cette dernière représente le résultat retourné par l'étape 1 de l'algorithme 23. Pour chaque arrêt $x_i \in P$, nous associons une fenêtre de temps $[t_a^r(x_i,p),\ t_d^r(x_i,p)]$ qui est composée respectivement de l'heure d'arrivée à l'arrêt x_i et l'heure à laquelle le passager quitte l'arrêt x_i.
Dans cet exemple, le chemin déterminé par l'API est composé de trois modes différents. Le passager attend à son point de départ trois minutes avant de monter dans le bus à 9h03, puis il est déposé à l'arrêt x_2. Il marche pendant 5 minutes depuis l'arrêt x_2 vers l'arrêt x_3 puis il attend 5 minutes avant

de prendre finalement le train pour rejoindre sa destination finale D_r (voir la figure 6.1).

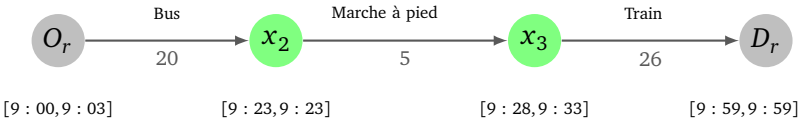


FIGURE 6.1 – Chemin multimodal classique P.

La figure 6.2 représente l'étape 2 de l'algorithme 23, où les chemins potentiels de substitution sont déterminés.

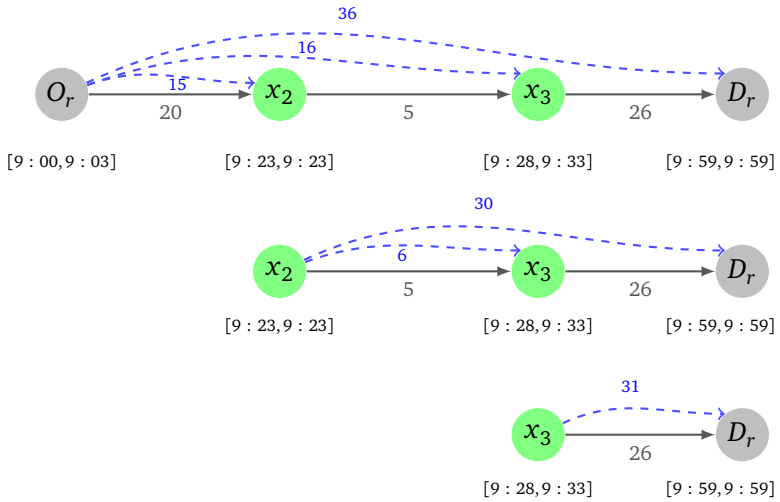


FIGURE 6.2 – Les plus courts chemins entre chaque paire (x_i, x_j) tel que l'arrêt x_j est situé après l'arrêt x_i sont représentés par des lignes en pointillé de couleur bleu.

Dans la figure 6.3, nous présentons les chemins potentiels de substitution retournés par l'algorithme 23. Le chemin (x_2, x_3) se traduit par une arrivée à x_3 plus tard que si on utilisait les transports publics, car il nécessite 6 minutes de x_2 à x_3, au lieu de 5 minutes. Une situation similaire se produit pour le chemin (x_3, D_r) : 31 minutes en covoiturage contre 26 minutes avec les transports publics. Donc les deux chemins (x_2, x_3) et (x_3, D_r) sont annulés.

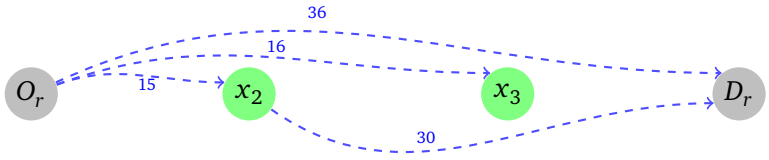


FIGURE 6.3 – Chemins potentiels pour une substitution.

6.3.2 Conducteurs potentiels

A ce niveau, nous définissons une estimation sur la proximité des chemins potentiels de substitution par rapport aux conducteurs. Cette estimation est définie comme une somme minimale de la distance estimée à partir de la position d'un conducteur vers un arrêt du chemin classique P, et inversement à partir de P vers la destination du conducteur. Quatre points différents sont utilisés de sorte que seules les substitutions potentielles sont autorisées. Pour ce faire, nous estimons la distance entre deux points donnés par leur latitude/longitude avec la formule de Haversine.

Algorithme 24 Proximité des conducteurs par rapport aux arrêts du chemin P

Entrée: Chemin multimodal classique P, liste des chemins potentiels $PDrive$.
Sortie: $LP_x^{\downarrow}$: Conducteurs potentiels pour une prise en charge à l'arrêt x.
$LP_y^{\uparrow}$: Conducteur potentiels pour une dépose à l'arrêt y.
LD_k : Ensemble des chemins potentiels pour le conducteur k.
Mise à jour de $PDrive$.
$LDriver$: Liste des conducteurs potentiels.

1: **Initialisation** : $LP_x^{\downarrow} \leftarrow \emptyset$, $LP_y^{\uparrow} \leftarrow \emptyset$, $PDrive' \leftarrow PDrive$
2: **Pour tout** conducteur k **Faire**
3: $LD_k \leftarrow \emptyset$.
4: **Pour tout** $(x, y) \in PDrive$ **Faire**
5: Estimer $\hat{\delta}(O_k, x)$ de O_k à x en utilisant la formule de Haversine.
6: Estimer $\hat{\delta}(y, D_k)$ de y à D_k en utilisant la formule de Haversine.
7: **Si** $\hat{\delta}(O_k, x) + \delta(x, y) + \hat{\delta}(y, D_k) \leq \lambda_k \delta(O_k, D_k)$ **Alors**
8: $LP_x^{\downarrow} \leftarrow LP_x^{\downarrow} \cup \{k\}$.
9: $LP_y^{\uparrow} \leftarrow LP_y^{\uparrow} \cup \{k\}$.
10: $LD_k \leftarrow LD_k \cup \{(x, y)\}$.
11: $PDrive' \leftarrow PDrive' \cup (x, y)$.
12: $LDriver \leftarrow LDriver \cup \{k\}$.
13: **Fin Si**
14: **Fin Pour**
15: **Fin Pour**
16: Mettre à jour l'ensemble $PDrive$ à $PDrive'$ (c'est à dire retirer de l'ensemble $PDrive$ tous les chemins de substitution (x, y) qui ne sont détectés par aucun conducteur potentiel).

L'algorithme 24 restreint la liste des conducteurs potentiels à ceux dont la position de départ et d'arrivée sont assez proche des arrêts de P et dont le temps de détour est inférieur à une valeur fixée pour les conducteurs correspondants. L'étape 7 vérifie l'admissibilité de la contrainte du temps de détour en déviant leur trajet via $x \rightarrow y$. Cette dernière est basée sur des estimations pour les durées manquantes de O_k vers x (étape 5) et de y vers D_k (étape 6). Pour chaque arrêt potentiel x pour une prise en charge, nous gardons seulement dans l'ensemble $LP_x^{\downarrow}$ les conducteurs qui respectent la borne supérieure sur la contrainte du temps de détour. La même méthode s'applique pour le lieu de dépose dans l'ensemble $LP_y^{\uparrow}$. Le détour maximal d'un conducteur ne doit pas dépasser 20%[27] de sa durée initiale (c'est à dire $\lambda_k = 1.2$). L'étape 2 parcourt l'ensemble des conducteurs dans le système ayant une même date de départ que l'utilisateur r et qui se situent également dans une même zone géographique.

27. En ayant un détour de moins de 20 %, les organismes d'assurances couvrent ce détour en cas d'accident de trajet. L'accident de mission est celui qui se produit pendant un déplacement ou une mission lors de laquelle le salarié est sous la subordination de l'employeur.

La figure 6.4 illustre la situation d'un conducteur k au moment de l'arrivée de la requête r.

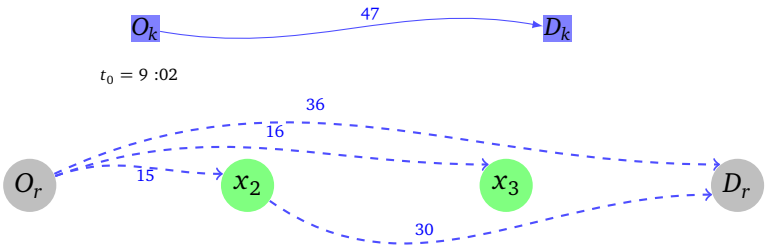


FIGURE 6.4 – Le conducteur k se déplace de O_k à D_k à l'instant 9h02. Les arrêts constituant le chemin P ainsi que les chemins potentiels de substitution sont représentés par les lignes en pointillé de couleur bleu.

La figure 6.5 montre les résultats de l'estimation en utilisant la formule Haversine (étapes 5 et 6).

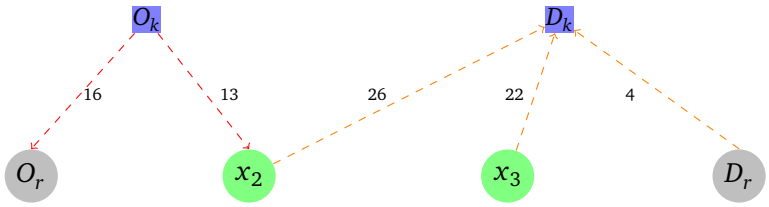


FIGURE 6.5 – Estimation de la durée vers chaque nœud potentiel de prise en charge depuis la position du conducteur, ainsi que la durée depuis chaque nœud potentiel de dépose vers la destination en utilisant la formule Haversine. Chaque position géographique est supposée être connue à l'instant de la requête.

La figure 6.6 représente les tests effectués sur chaque chemin potentiel de substitution depuis la position de départ du conducteur jusqu'à sa destination. Les chiffres sur les arcs en pointillé représentent les durées estimées, tandis que les chiffres sur les arcs en lignes continues représentent les durées réelles (calculées par l'algorithme 23).

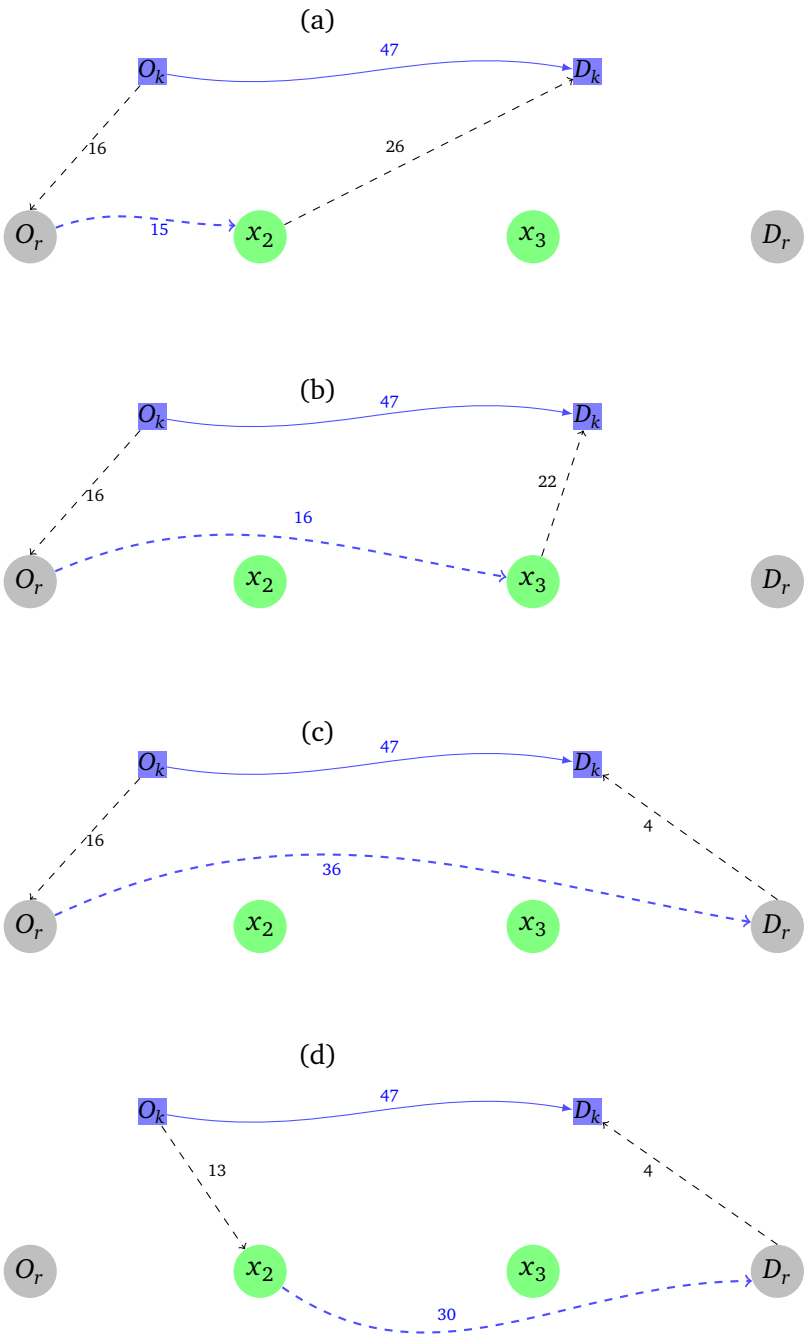


FIGURE 6.6 – Tous les chemins potentiels de substitution sont évalués par rapport à la contrainte du temps de détour du conducteur k.

La figure 6.7 détaille le résultat de l'algorithme 24, où l'arc (O_r, x_2) a été supprimé car la nouvelle durée $16 + 15 + 26 = 57$ minutes, dépasse le détour maximal fixé par le conducteur ($1,2 \times 47 = 56,4$ minutes). Ce dernier sera retiré de la liste des chemins potentiels de substitution LD_k pour le conducteur k. Cela permet également de restreindre la liste des lieux potentiels de prise en charge et de dépose.

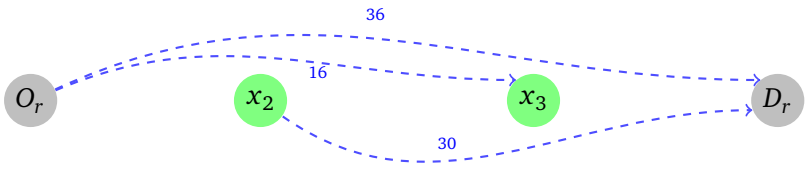


FIGURE 6.7 – Les chemins qui satisfont la borne supérieure sur la contrainte du temps de détour.

6.3.3 Calcul de plus courts chemins et de la meilleure substitution en covoiturage

Une fois que la liste des conducteurs potentiels est déterminée, l'algorithme 25 calcule l'ensemble des plus courts chemins afin de vérifier l'admissibilité des trois contraintes, à savoir : *le temps de détour du conducteur*, *le temps d'attente à l'emplacement de prise en charge* et le *temps d'arrivée du passager*. Deux types de calculs de plus courts chemins sont effectués. Le premier depuis la position actuelle de chaque conducteur vers chaque arrêt potentiel de prise en charge (étape 4). Le second, depuis chaque arrêt potentiel de dépose vers la destination de chaque conducteur (étape 10) (voir la figure 6.8).

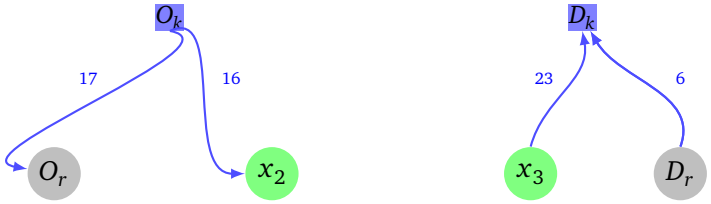


FIGURE 6.8 – Les plus courts chemins exacts sont représentés par des lignes continues de couleur bleu. L'ensemble des chemins calculés sont limités aux lieux potentiels de prise en charge et de dépose.

Ces calculs seront effectués en deux phases. La première phase détermine les durées depuis leur lieu de départ vers les lieux potentiels de prise en charge. En se basant sur ces premières durées calculées, nous mettons à jour l'ensemble $LP_y^{\uparrow}$ en enlevant les conducteurs qui ne respectent pas une des contraintes : la contrainte du temps d'attente du conducteur, la contrainte du temps de détour ou la contrainte du temps d'arrivée du passager. Une fois que l'ensemble de dépose $LP_y^{\uparrow}$ est mis à jour, la deuxième phase calcule les durées exactes depuis chaque lieu de dépose y vers la destination de chaque conducteur contenue dans la liste $LP_y^{\uparrow}$. Dans la seconde phase, à chaque fois qu'une durée d'un conducteur k est calculée, nous parcourons l'ensemble de ses trajets potentiels LD_k en vérifiant l'admissibilité exacte de la contrainte du temps de détour du conducteur. Enfin, parmi tous les trajets admissibles, nous sélectionnons celui qui génère le plus d'économies en temps d'arrivée.

L'objectif est donc de trouver un moyen de se rendre à des points de correspondance en utilisant le covoiturage comme mode de transport. Autrement dit, nous sélectionnons le *meilleur chemin de substitution* $(x^\star, y^\star)$ qui génère un gain en temps d'arrivée le plus élevé au passager si ce dernier

partage son trajet avec le conducteur $k^\star$ depuis $x^\star$ jusqu'à l'arrêt $y^\star$ plutôt que les transports publics (voir la figure 6.9). Pour chaque gain positif généré autour de chaque arrêt potentiel de dépose, un itinéraire multimodal classique est lancé en prenant en compte la nouvelle heure d'arrivée du passager à l'arrêt concerné. Ceci permet de garantir au passager une arrivée plus tôt que celle annoncée par l'API (étapes 17 à 23 de l'algorithme 25).

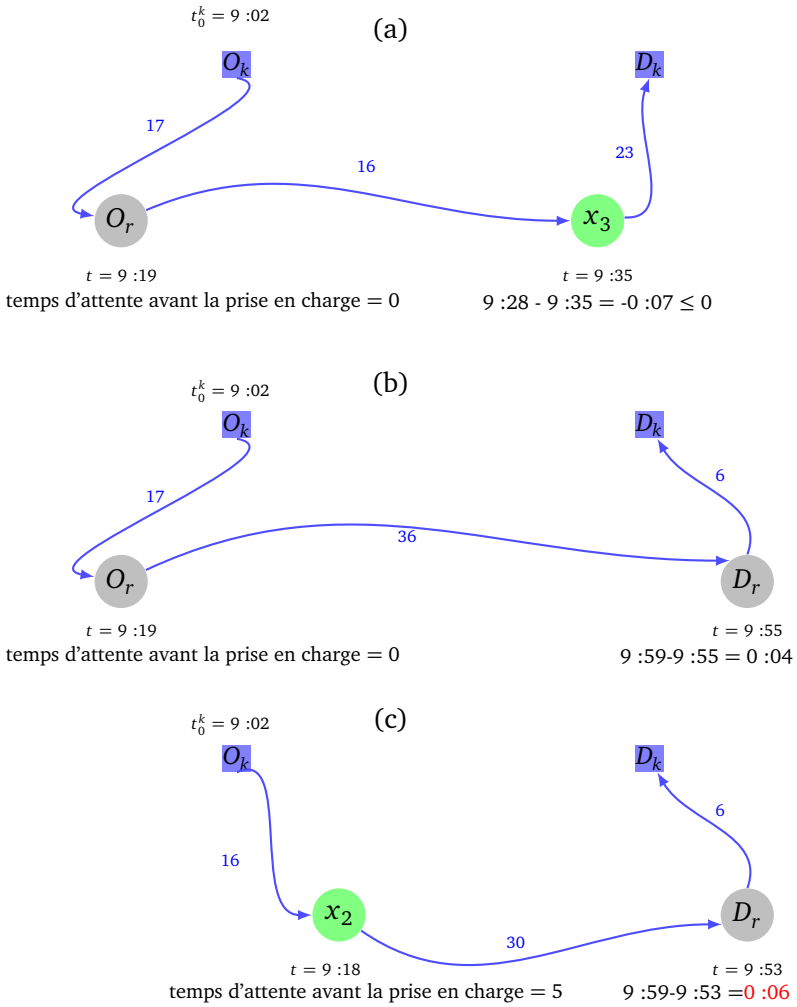


FIGURE 6.9 – L'heure d'arrivée en covoiturage à x_3 est plus tardive que l'heure d'arrivée en transport public ; ce covoiturage est donc retiré. Entre les deux chemins (O_r, D_r) et (x_2, D_r), le deuxième améliore l'heure d'arrivée à D_r, il est donc le meilleur chemin de substitution.

Algorithme 25 Plus courts chemins et détermination de la meilleure substitution

Entrée: Demande $O_r D_r$,
Transports publics (API)
$LP_x^{\uparrow}$: Liste de conducteurs potentiels pour une prise en charge à l'arrêt x.
$LP_y^{\downarrow}$: Liste de conducteurs potentiels pour une dépose à l'arrêt y.
LD_k : Ensemble des chemins potentiels pour le conducteur k. $k \in LDriver$.
Sortie: Meilleur conducteur $k^\star$
Meilleur chemin de substitution $(x^\star, y^\star)$ ou échec de substitution

Initialisation : $optimal\_arrival\_time \leftarrow t_a^u(x,p)$, $k' \leftarrow null$
Pour tout $x \in P$ **Faire**
  Si $LP_x^{\uparrow}$ est non vide **Alors**
    Calculer les plus courts chemins depuis $LP_x^{\uparrow}$ vers x en utilisant l'algorithme de Dijkstra inverse plusieurs-vers-un.
  Fin Si
  Mise à jour de l'ensemble $LP_y^{\uparrow}$, $y > x$, en se basant sur les durées exactes calculées dans l'étape 4.
Fin Pour
Pour tout $y \in P$ **Faire**
  Si $LP_y^{\downarrow}$ est non vide **Alors**
    Calculer les plus courts chemins depuis $LP_y^{\downarrow}$ vers y en utilisant l'algorithme de Dijkstra un-vers-plusieurs.
    Pour tout conducteur k dans $LP_y^{\downarrow}$ **Faire**
      Récupérer la liste des chemins potentiels LD_k pour le conducteur k
      Vérifier l'admissibilité entre le conducteur k et l'utilisateur r pour les chemins potentiels $x \rightarrow y, \forall x < y$.
      Garder le meilleur conducteur k' et le chemin $(x^\star, y)$ qui génèrent le gain en temps d'arrivée le plus élevé, c'est à dire, $\max_{k',x^\star,y} \left\{ t_a^u(y,p) - (\max\{t_a^{k'}(x^\star,c), t_a^u(x^\star,p) + \tau(x^\star)_{pc}\} + \delta(x^\star,y)) \right\}$ ainsi que son heure d'arrivée à y, c'est à dire $t_a^\star(y) \leftarrow \max\{t_a^{k'}(x^\star,c), t_a^u(x^\star,p) + \tau(x^\star)_{pc}\} + \delta(x^\star,y)$
    Fin Pour
  Fin Si
  Si k' est non nul **Alors**
    Calculer un plus court chemin multimodal classique $P = y, \ldots, x_{nbs}, D_r$ en utilisant l'API, partant de y à l'instant $t_a^\star(y)$, avec l'heure d'arrivée à la destination $arrival\_time\_at\_D_r$.
    Si $arrival\_time\_at\_D_r < optimal\_arrival\_time$ **Alors**
      $optimal\_arrival\_time \leftarrow arrival\_time\_at\_D_r$
      $k^\star \leftarrow k'$, $x^\star \leftarrow x$, $y^\star \leftarrow y$
    Fin Si
  Fin Si
  $k' \leftarrow null$
Fin Pour
Retourner meilleur conducteur k', meilleur arrêt de prise en charge $x^\star$ et meilleur arrêt de dépose $y^\star$

L'algorithme 26 décrit le processus global de l'approche. L'étape 7 est effectuée une fois que le passager atteint le lieu de dépose $y^\star$. Cependant, cette dernière n'est pas comprise dans le processus d'exécution de la requête.

Algorithme 26 Processus global de la première approche

Entrée: Demande $O_r D_r$, instant de départ t_r
Sortie: Meilleure substitution ou échec de substitution.

1: Déterminer un plus court chemin multimodal classique partant de O_r vers D_r à l'instant t_r ainsi que les chemins potentiels de substitution en utilisant l'**Algorithme 23**
2: Sélectionner les conducteurs potentiels en utilisant l'**Algorithme 24**
3: Déterminer la meilleure substitution $(x^\star, y^\star)$ ainsi que le meilleur conducteur associé $k^\star$ en utilisant l'**Algorithme 25**
4: **Si** $y^\star \neq D_r$ **Alors**
5: $\quad t'_r \leftarrow \max\{t_a^{k^\star}(x^\star, c), t_a^r(x^\star, p) + \tau(x^\star)_{pc}\} + \delta(x^\star, y^\star)$
6: $\quad$ Mise à jour de la route du passager $O_{r^\star} \rightsquigarrow x^\star \rightsquigarrow y^\star$.
7: $\quad$ Remonter à l'étape 1 en prenant $y^\star$ comme point de départ et t'_r comme l'instant de départ.
8: **Fin Si**

Dans notre exemple, l'emplacement de dépose correspond à la destination du passager. Ainsi, le nouvel itinéraire du passager $(O_r \rightsquigarrow x_2 \rightsquigarrow D_r)$ est représenté dans la figure 6.10.

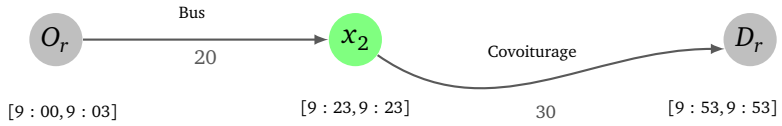


FIGURE 6.10 – Nouvel itinéraire du passager.

6.3.4 Complexité de l'approche directe

Nous décrivons la complexité de l'ensemble du processus. L'algorithme 23 initialise le traitement de la requête en $O(\mathscr{D}_{road} \cdot (|P| - 1) + \mathscr{D}_{API})$, où $\mathscr{D}_{road}$ (resp. $\mathscr{D}_{API}$) est la complexité du Dijkstra dans un réseau routier (resp. réseau multimodal). L'algorithme 24 permet d'estimer la proximité des conducteurs par rapport aux chemins potentiels de substitution en $O(|LDriver| \cdot |PDrive|)$. Sur l'ensemble des conducteurs potentiels sélectionnés, l'algorithme 25 vérifie leurs admissibilités en termes de contraintes de détour en déterminant la meilleure substitution. Ce dernier est donné en $O(|P^+| \cdot (\mathscr{D}_{road}) + \sum_{x \in P^+} \sum_{y>x} |LP_y^{\uparrow}| + |P^-| \cdot \mathscr{D}_{road} + \sum_{y \in P^-} |LP_y^{\downarrow}|)$, où $|P^+|$ et $|P^-|$ sont respectivement le nombre d'arrêts potentiels de prise en charge et de dépose. Enfin, la complexité de l'approche globale est de $O\big((|P^+|+1) \cdot \mathscr{D}_{API} + |P^+ + P^- + P| \cdot \mathscr{D}_{road} + \sum_{x \in P^+} \sum_{y>x} |LP_y^{\uparrow}| + \sum_{y \in P^-} |LP_y^{\downarrow}| + |LDriver| \cdot |PDrive|\big)$. Le processus se ré-exécute une fois que le passager atteint son lieu de dépose $y^\star$ en prenant en compte sa nouvelle heure de départ.

6.4 Deuxième approche : Approche par étiquetage

L'idée principale de cette approche consiste à étiqueter chaque itinéraire d'un conducteur sur une carte géographique. Cette dernière nécessite la présence d'une carte géographique afin de géolocaliser et d'étiqueter en temps réel les positions actuelles de chaque conducteur.

6.4.1 Étape d'initialisation

La première étape est assez similaire que celle décrite dans 6.3.1. Plus spécifiquement, lorsqu'une requête O_rD_r arrive dans le système à l'instant t_r, un plus court chemin multimodal classique ainsi que les chemins possibles en covoiturage entre les différents arrêts de correspondance sont déterminés. Nous reprenons le même exemple présenté dans l'approche précédente avec quelques modifications sur les durées des sous-trajets. Sur l'itinéraire récupéré, nous associons une heure d'arrivée du passager à chaque arrêt constituant ce chemin (voir figure 6.11).

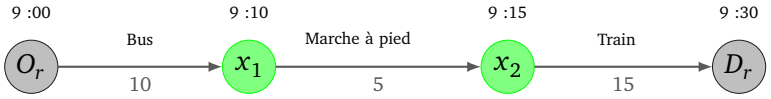


FIGURE 6.11 – Itinéraire multimodal classique composé de deux correspondances.

Ensuite, pour chaque arrêt $x_i \in P/\{D_r\}$, nous calculons les plus courtes durées pour atteindre chaque arrêt x_j tel que x_j est situé après l'arrêt de x_i dans le chemin P. Ces durées sont ensuite utilisées pour tester la potentialité des chemins pour des éventuelles substitutions. A cet effet, nous ne conservons que les chemins qui permettent de diminuer l'heure d'arrivée du passager aux arrêts constituant son trajet.
Contrairement à l'étape d'initialisation de la première approche décrite dans la partie 6.3.1, nous associons une fenêtre de temps de prise en charge $[d_{\min}^{x_i}, d_{\max}^{x_i}]$ pour chaque arrêt potentiel x_i. Cette dernière permet de délimiter la période de temps dans laquelle le passager peut être pris en charge tout en assurant une heure d'arrivée plus tôt que celle annoncée par le chemin multimodal classique. Dans ce cas, $d_{\min}^{x_i}$ correspond à l'heure d'arrivée du passager en transport public à x_i c'est à dire $t_a^r(x_i, p)$, tandis que $d_{\max}^{x_i}$ est l'heure de départ au plus tard dans laquelle le passager doit quitter l'arrêt x_i pour une arrivée plus tôt vers les autres arrêts potentiels. En conséquent, cela permet d'assurer au passager ayant une prise en charge à x_i dans la fenêtre de temps adéquate $[d_{\min}^{x_i}, d_{\max}^{x_i}]$, une heure de dépose à un autre arrêt x_j plus tôt que s'il utilisait les transports publics. La figure 6.12 représente les fenêtres de temps associées à chaque arrêt potentiel de prise en charge. Les calculs effectués au dessus des nœuds déterminent les heures de départ au plus tard, de sorte que l'heure d'arrivée à x_j est inférieure ou égale à l'heure d'arrivée en utilisant les transports publics. Finalement, chaque fenêtre de temps est composée d'une heure de départ au plus tôt et au plus tard pour chaque arrêt potentiel.

Pour chaque arrêt x_i, nous associons une liste atteignable d'arrêts $PDrive(x_i)$. Cette dernière contient les arrêts $x_j \in P$ tel que $t_a^r(x_i, p) + \tau(x_i)_{pc} + \delta(x_i, x_j) \leq t_a^r(x_j, p)$. Cette contrainte assure que l'heure d'arrivée en covoiturage à l'arrêt x_j venant de l'arrêt x_i (à l'instant $t_a^r(x_i, p)$) est plus tôt que l'heure d'arrivée à x_j prévue par l'API. Le temps nécessaire pour changer de modalité à l'arrêt de prise en charge x_i des transports publics vers le covoiturage (c'est à dire $t_a^r(x_i, p) + \tau(x_i)_{pc} + \delta(x_i, x_j)$) est pris en compte.

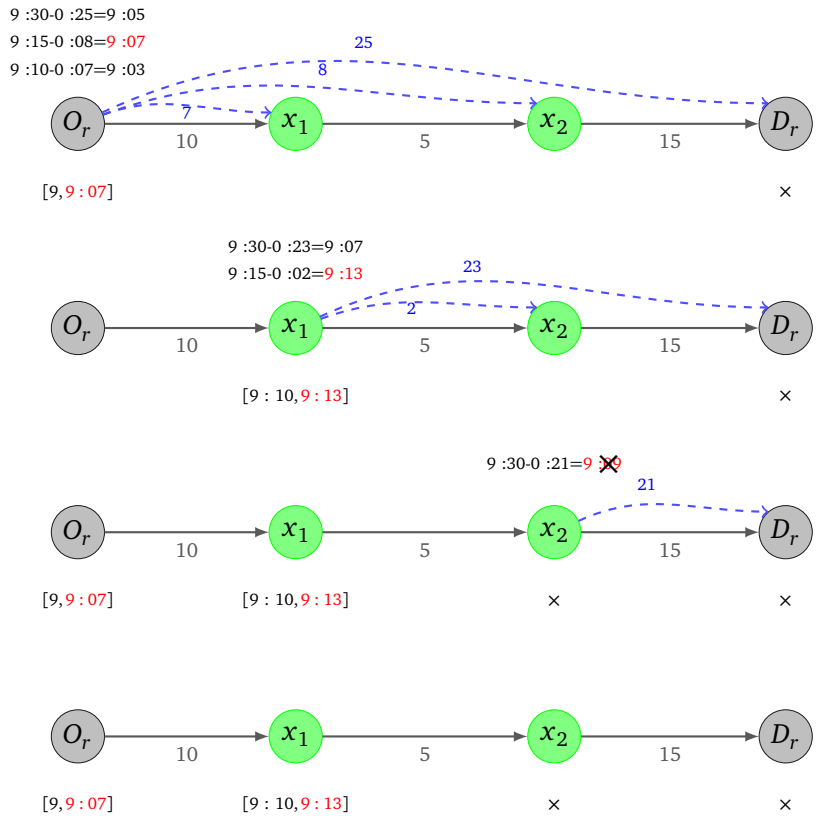


FIGURE 6.12 – Détermination des fenêtres de temps de prise en charge pour chaque arrêt potentiel. L'arrêt x_2 ne comporte pas de fenêtre de temps car il est impossible de réduire le temps d'arrivée en covoiturage vers d'autres arrêts en ayant x_2 comme lieu de prise en charge.

La figure 6.13 montre les chemins potentiels en covoiturage.

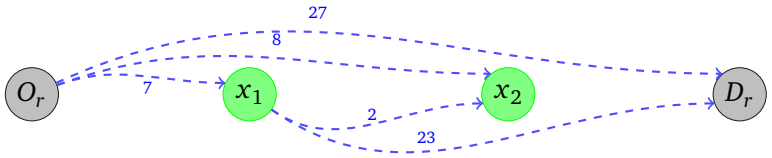


FIGURE 6.13 – Chemins potentiels de substitution.

L'algorithme 27 décrit ce processus d'initialisation.

Algorithme 27 Initialisation de la requête

Entrée: Demande $O_r D_r$, instant de départ t_r
Sortie: Chemin multimodal classique P,
Fenêtre de temps de prise en charge $[d_{\min}^{x_i}, d_{\max}^{x_i}], \forall x_i \in P$,
Ensemble de chemins potentiels de substitution $PDrive(x_i), \forall x_i \in P$.

1: Calculer un plus court chemin multimodal classique $P = O_r, x_1, \ldots, x_{nbs}, D_r$ en utilisant l'API, partant de O_r à l'instant t_r, avec l'heure d'arrivée $t_a^r(x_i, p)$, $x_i \in P$.
2: Calculer les plus courts chemins entre les différents arrêts P on associant le temps d'arrivée $t_a^r(x_i, c)$ à chaque arrêt x_i, $x_i \in P$.
3: Pour chaque arrêt x_i, déterminer la fenêtre de temps de prise en charge $[d_{\min}^{x_i}, d_{\max}^{x_i}]$ dans laquelle le passager peut être pris en charge par les conducteurs, telle que

$$d_{\min}^{x_i} \leftarrow t_a^r(x_i, p) + \tau(x_i)_{pc}$$

$$d_{\max}^{x_i} \leftarrow \max_{x_j \in P_i^{'}} \Big\{ \Delta^{x_i, x_j}, \text{ où } \Delta^{x_i, x_j} = t_a^r(x_j, p) - (\tau(x_i)_{pc} + \delta(x_i, x_j)) \text{ et } P_i^{'} = \{x_j \in P \text{ telle que } t_a^r(x_i, p) + \tau(x_i)_{pc} + \delta(x_i, x_j) \leq t_a^r(x_j, p)\} \Big\}$$

4: Pour chaque arrêt x_i, garder dans la liste $PDrive(x_i)$ uniquement les chemins potentiels qui respectent la contrainte $t_a^r(x_i, p) + \tau(x_i)_{pc} + \delta(x_i, x_j) \leq t_a^r(x_j, p)$.

6.4.2 Conducteurs potentiels

A ce niveau, l'ensemble des chemins potentiels de substitution ainsi que les fenêtres de temps de prise en charge des arrêts potentiels sont déjà déterminés. La seconde étape consiste à déterminer la liste des conducteurs potentiels capables de prendre en charge le passager sur les différents arrêts possibles tout en respectant les fenêtres de temps de prise en charge associées. Cette liste est également restreinte en fonction du *temps d'attente maximal* ainsi que le *temps de détour maximal* des conducteurs. Les plus courts chemins depuis les positions actuelles de ces conducteurs vers les arrêts potentiels de prise en charge sont déterminés pendant l'exploration de cet espace de recherche. Une des manières de déterminer la liste des conducteurs potentiels consiste à calculer pour chaque conducteur son plus court chemin depuis sa position actuelle vers les arrêts potentiels de prise en charge, en gardant uniquement les conducteurs qui respectent la fenêtre de temps de prise en charge ainsi que la contrainte du temps de détour maximal. Dans notre approche, nous utilisons un seul algorithme de plus court chemin de type un-vers-plusieurs, calculé inversement depuis chaque arrêt potentiel de prise en charge.
Un sous-problème à soulever à ce niveau est la sélection des conducteurs dont leur temps d'arrivée à x_i se trouve dans la fenêtre de temps $[d_{\min}^{x_i} - \theta, d_{\max}^{x_i}]$. La durée θ ajuste la fenêtre de temps de prise en charge pour ne pas exclure les conducteurs qui arrivent plus tôt que l'heure d'arrivée du passager à x_i (c'est à dire $d_{\min}^{x_i}$). En effet, le temps d'attente d'un conducteur dans ce cas là ne doit pas dépasser son temps maximal fixé. Une borne inférieure triviale de θ correspond à la plus grande durée d'attente maximale fixée parmi tous les conducteurs se trouvant dans le système (c'est à dire $\theta = \max_{k \in database}\{w_{\max}^k\}$). Malheureusement, la détermination de cette dernière nécessiterai le parcours de tous les conducteurs existant dans le système. Pour éviter ceci, nous fixons θ à $(t_a^r(x_i, p) - t_r)$, qui correspond à la durée restante pour que le passager puisse atteindre x_i depuis son origine O_r en transport public. De cette manière, nous évitons l'élimination de certains conducteurs potentiels. Notre idée est cependant basée sur la durée maximale pour atteindre un des arrêts potentiels de prise en charge depuis leur position actuelle dans laquelle la substitution n'est plus permise. Cette durée maximale correspond à la somme de la durée en transport public entre O_r et

x_i, c'est à dire $\theta = (t_a^r(x_i, p) - t_r)$ plus la longueur de la fenêtre de temps x_i, c'est à dire $(d_{\max}^{x_i} - d_{\min}^{x_i})$. Nous notons cette quantité par $Max\_Duration(x_i)$, qui n'est autre que la durée maximale tolérée avant la prise en charge à x_i. La figure 6.14 représente l'espace de recherche autour de l'arrêt O_r, limité par $Max\_Duration(O_r) = t_a^r(O_r, p) - t_u + (d_{\max}^{O_r} - d_{\min}^{O_r}) = 7$ minutes.

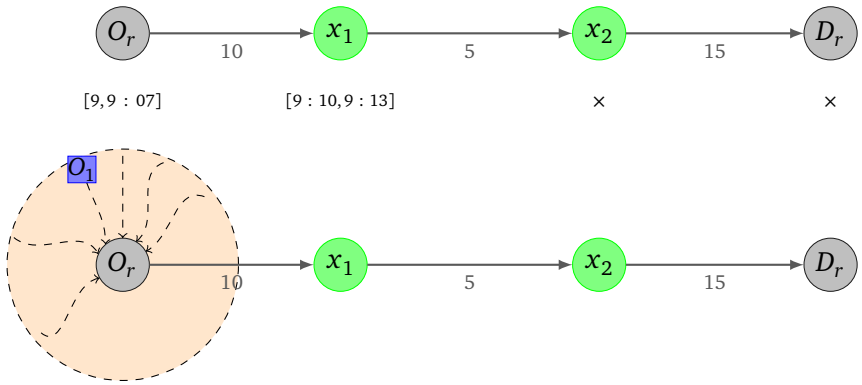


FIGURE 6.14 – Espace de recherche autour de l'arrêt O_r. Ce dernier est limité à 7 minutes, de telle sorte que l'heure de prise en charge ne dépasse pas 9 : 07.

Lorsqu'un conducteur est marqué pendant la procédure de recherche autour d'un arrêt x_i, quelques tests doivent être effectués. Plus spécifiquement, nous vérifions l'admissibilité de la contrainte du temps de détour du conducteur sur l'ensemble des chemins potentiels $PDrive(x_i)$. Pour chaque arrêt x_j dans $PDrive(x_i)$ qui respecte la contrainte du temps de détour du conducteur, nous stockons la durée restante avant que sa contrainte de détour ne soit plus respectée ($Label(x_j)$). En plus de la durée restante du temps de détour, nous gardons également d'autres informations dans $DriverList(x_j)$, à savoir : l'identifiant du conducteur k ainsi que l'arrêt de prise en charge x_i. A ce niveau, les durées des trajets depuis l'ensemble des arrêts potentiels de dépose vers les destinations des conducteurs ne sont pas encore déterminées. Nous utilisons donc des durées estimées en utilisant la formule de Haversine (voir figure 6.15).

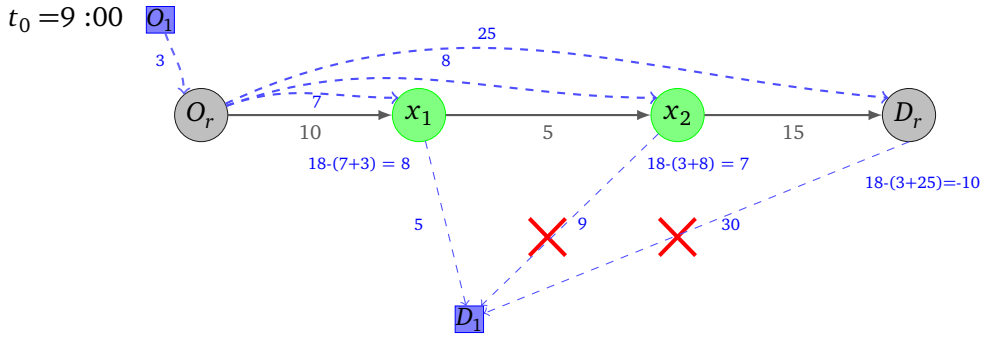


FIGURE 6.15 – Le conducteur partant de sa position actuelle O_1 à l'instant 9 : 00 peut atteindre le point de départ du passager O_r en seulement 3 minutes. Les durées du trajet de O_r vers x_1, x_2, D_r sont respectivement de 7, 8 et 25 minutes. En revanche, les durées estimées à partir des arrêts potentiels de dépose x_1, x_2 et D_r vers la destination du conducteur D_1 sont respectivement de 5, 9 et 30 minutes. Le coefficient de détour λ_1 est fixé à 1.2 et la durée directe de son trajet O_1D_1 est de 15 minutes. Les durées restantes du temps de détour depuis x_1, x_2, D_r jusqu'à la destination du conducteur D_1 sont respectivement de 8, 7 et -10. Ainsi, seul l'arrêt x_1 est susceptible de respecter la contrainte de détour. Finalement, l'étiquette associée à l'ensemble $PDrive(x_1)$ est (Conducteur 1 ;x_1 ;8).

Cette procédure est répétée pour chaque arrêt dont la fenêtre de temps est non-vide. A chaque arrêt potentiel de dépose x_j, nous mettons à jour l'étiquette $Label(x_j)$ à la plus grande durée restante de détour (voir la figure 6.16)

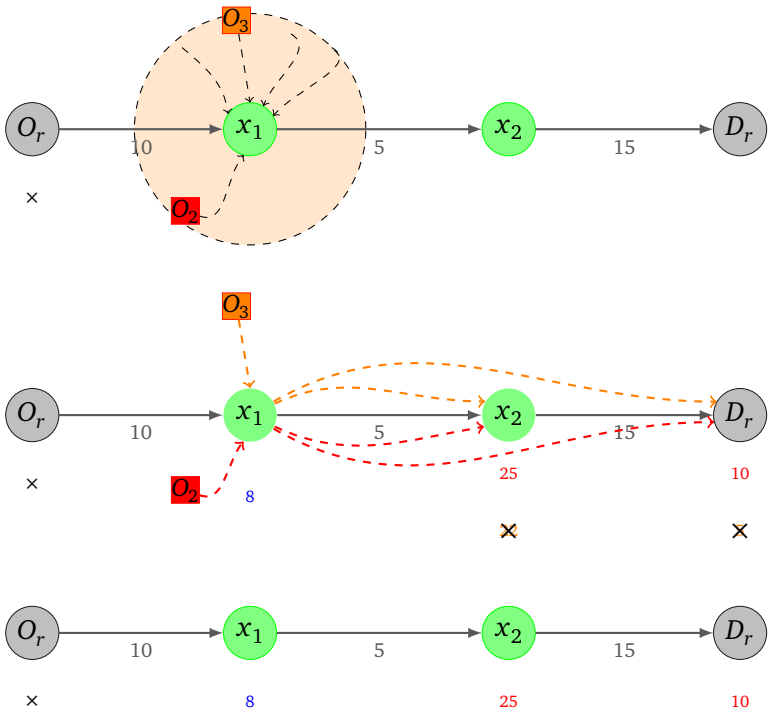


FIGURE 6.16 – Au cours de la procédure de recherche autour de l'arrêt x_1, deux conducteurs sont détectés. Les étiquettes du temps de détour restant associées aux arrêts x_2 et D_r sont respectivement mis à jour à 25 et à 10 minutes.

L'algorithme 28 définit ce processus.

Algorithme 28 Conducteurs potentiels

Entrée: Demande $O_r D_r$, instant de départ t_r
Fenêtre de temps de prise en charge $[d_{\min}^{x_i}, d_{\max}^{x_i}], \forall x_i \in P$.
$PDrive(x_i)$: liste des chemins potentiels ayant x_i comme arrêt de prise en charge.$\forall x_i \in P$.
Sortie: $DriverList(x_i)$: liste des conducteurs passant par l'arrêt x_i .$\forall x_i \in P$.
$Label(x_i)$: la plus grande durée restante de détour autour de l'arrêt x_i , $\forall x_i \in P$.

1: **Initialisation** : $Label(x_i) \leftarrow 0, \forall x_i \in P$.
2: **Pour tout** $x_i \in P$, avec une fenêtre de temps non vide **Faire**
3: $Max\_Duration(x_i) \leftarrow (t_a^r(x_i, p) - t_r) + (d_{\max}^{x_i} - d_{\min}^{x_i})$.
4: Calculer l'algorithme de Dijkstra inverse un-vers-plusieurs depuis x_i limité par la durée $Max\_Duration(x_i)$
5: **Pour tout** nœud marqué O_k correspondant à une des positions actuelles des conducteurs **Faire**
6: **Pour tout** x_j **dans** $LDriver(x_i)$ **Faire**
7: Estimer $\hat{\delta}(x_j, D_k)$ de x_j vers D_k en utilisant la formule de Haversine
8: **Si** (x_i, x_j) est un *chemin admissible* pour le conducteur k et le passager r, en utilisant la durée estimée $\hat{\delta}(x_j, D_k)$ **Alors**
9: $DriverList(x_j) \leftarrow DriverList(x_j) \cup \{(k, O_k \rightsquigarrow x_i \rightsquigarrow x_j)\}$
10: $Remaining\_detour\_time \leftarrow \lambda_k \delta(O_k, D_k) - \big(\delta(O_k, x_i) + \max\{t_a(x_i, p) + \tau(x_i)_{pc} - t_a^k(x_i, c), 0\} + \delta(x_i, x_j)\big)$
11: **Si** $Label(x_j) < Remaining\_detour\_time$ **Alors**
12: $Label(x_j) \leftarrow Remaining\_detour\_time$
13: **Fin Si**
14: **Fin Si**
15: **Fin Pour**
16: **Fin Pour**
17: **Fin Pour**

6.4.3 Conducteurs admissibles

Les étiquettes associées à chaque arrêt potentiel de dépose x_j (c'est à dire $Label(x_j)$) permettent de limiter l'espace de recherche autour des arrêts potentiels de dépose. Pour ce faire, nous utilisons l'algorithme de Dijkstra pour chaque arrêt potentiel de dépose x_j dont lequel la recherche associée est limitée par la durée $Label(x_j)$. Pour chaque nœud marqué correspondant à une des destinations d'un conducteur et qui se trouve également dans la liste $PDrive(x_j)$, nous obtenons l'admissibilité exacte de la contrainte du temps de détour du conducteur. La figure 6.17 montre l'espace de recherche autour des arrêts x_1 et x_2 qui sont limités respectivement par une durée de 7 et 2 minutes.

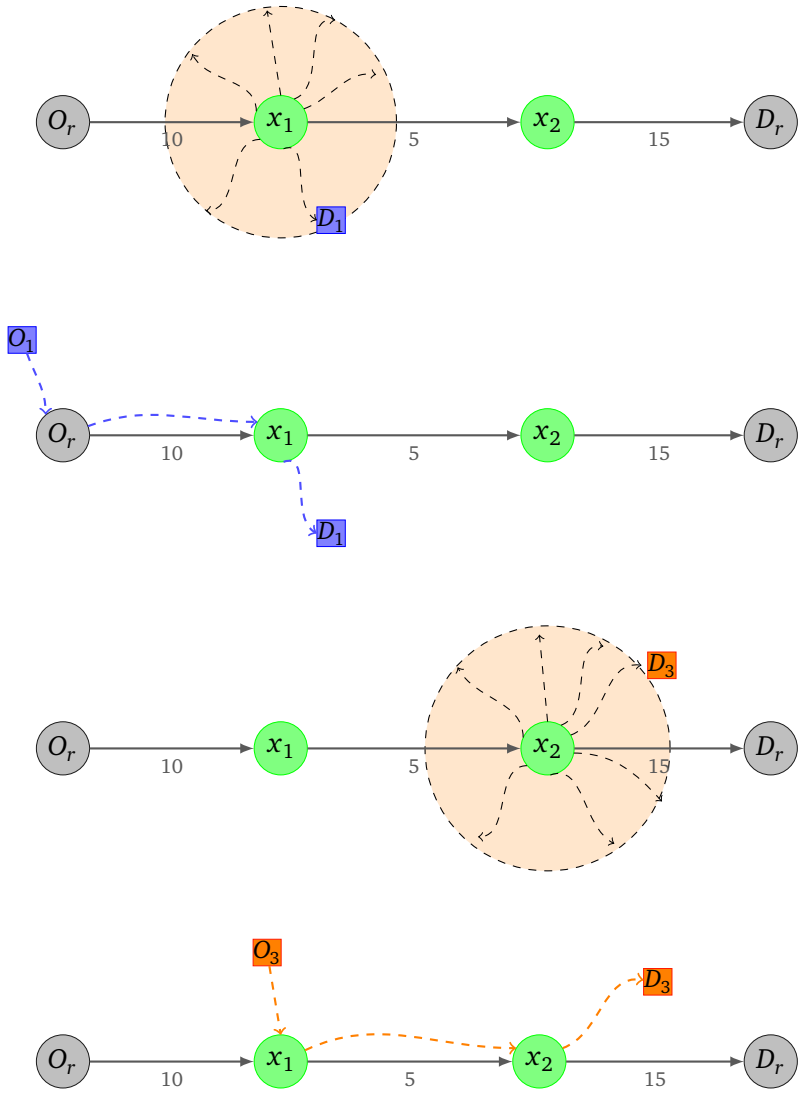


FIGURE 6.17 – A l'intérieur de l'espace de recherche de x_1 et x_2, deux chemins admissibles sont détectés. Le premier ayant x_1 comme lieu de dépose (c'est à dire $O_1 \rightsquigarrow O_r \rightsquigarrow x_1 \rightsquigarrow D_1$) et le second chemin ayant x_2 comme lieu de dépose (c'est à dire $O_3 \rightsquigarrow x_1 \rightsquigarrow x_2 \rightsquigarrow D_3$).

Dans le cas où plusieurs conducteurs ont été marqués autour d'un arrêt de dépose, on gardera celui qui emmène le passager le plus tôt possible à l'arrêt concerné (voir figure 6.18).

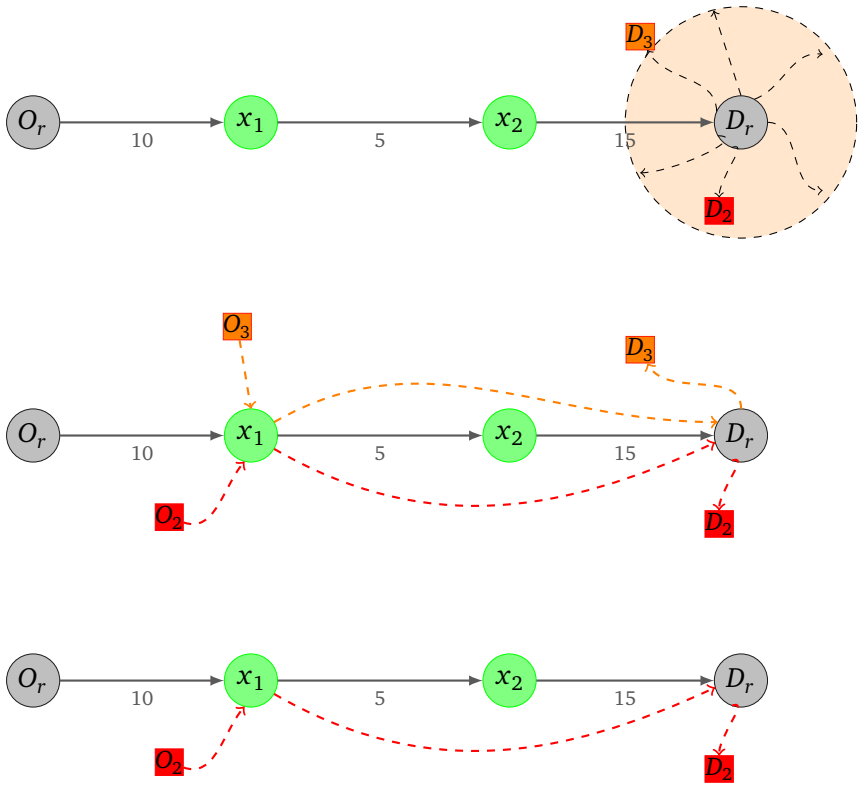


FIGURE 6.18 – A l'intérieur de l'espace de recherche autour de D_r, deux chemins admissibles sont détectés. Seulement le chemin avec une arrivée au plus tôt à D_r est maintenu.

Ainsi, pour chaque arrêt potentiel de dépose x_j, nous avons associé un meilleur conducteur qui permet au passager une arrivée plus tôt que celle du transport public. Ces conducteurs génèrent une économie en temps pour le passager en utilisant le covoiturage plutôt que les transports publics.

Le temps d'arrivée au plus tôt à l'arrêt x_j et le conducteur associé sont respectivement stockés dans $t_a^\star(x_j)$ et $Driver\_stop(x_j)$ tel que décrit dans l'algorithme 29. Le meilleur *chemin de substitution* pour chaque arrêt potentiel de dépose est présenté dans la figure **??**.

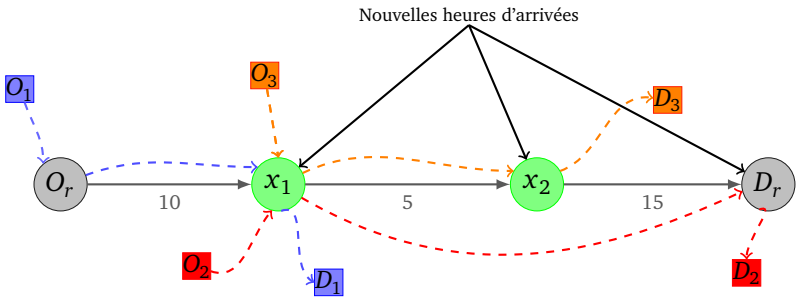


FIGURE 6.19 – Nouvelles heures d'arrivées vers les lieux potentiels de dépose.

Algorithme 29 Conducteurs admissibles

Entrée: Demande $O_r D_r$, instant de départ t_r
$Label(x_i)$: la plus grande durée parmi les durées restantes de détour autour de x_i, $\forall x_i \in P$
$DriverList(x_i)$: liste des conducteurs potentiels passant par via x_i,$\forall x_i \in P$.
Sortie: $t_a^\star(x_i)$: l'heure d'arrivée au plus tôt du passager à l'arrêt x_i, $\forall x_i \in P$.
$\sigma^\star(x_i)$: le temps de réduction généré en utilisant le covoiturage vers l'arrêt x_i, $\forall x_i \in P$.
$Driver\_stop(x_i)$: le meilleur conducteur associé à l'arrêt x_i, $\forall x_i \in P$.

1: **Pour tout** $x_j \in P$, avec $Label(x_j) > 0$ **Faire**
2: **Initialisation** : $\sigma^\star \leftarrow 0$, $k^\star \leftarrow$ -1
3: Calculer l'algorithme de Dijkstra inverse un-vers-plusieurs depuis x_j limité par la durée $Label(x_j)$
4: **Pour tout** nœud marqué D_k correspondant à une destination d'un conducteur dans $DriverList(x_j)$ **Faire**
5: **Pour tout** $(O_k \rightsquigarrow x_i \rightsquigarrow x_j) \in DriverList(x_j)$ **Faire**
6: **Si** (x_i, x_j) est un *chemin de substitution admissible* pour le conducteur k et le passager r **Alors**
7: $\sigma \leftarrow t_a^r(x_j,p) - \big(t_a^k(x_i,c) + \max\{t_a^k(x_i,c) - (t_a^r(x_i,p) + \tau(x_i)_{pc}), 0\} + \tau(x_i)_{pc} + \delta(x_i,x_j)\big)$
8: **Si** $\sigma^\star \leq \sigma$ **Alors**
9: $\sigma^\star \leftarrow \sigma$
10: $k^\star \leftarrow k$
11: $t_a^\star(x_j) \leftarrow t_a^k(x_i,c) + \max\{t_a^k(x_i,c) - (t_a^r(x_i,p) + \tau(x_i)_{pc}), 0\} + \tau(x_i)_{pc} + \delta(x_i,x_j)$
12: **Fin Si**
13: **Fin Si**
14: **Fin Pour**
15: **Fin Pour**
16: $Driver\_stop(x_j) \leftarrow k^\star$, $\sigma^\star(x_j) \leftarrow \sigma^\star$
17: **Fin Pour**

6.4.4 Meilleure substitution et mise à jour de l'itinéraire du passager

Afin de vérifier la potentialité des durées économisées sur chaque arrêt x_j, le plus court chemin en transport public est mis à jour à partir de x_j en prenant en compte la nouvelle heure d'arrivée $t_a^\star(x_j)$. Parmi tous les nouveaux chemins en transport public renvoyés par l'API, nous sélectionnons celui qui permet d'emmener le passager le plus tôt possible à sa destination finale D_r. Si aucun covoiturage n'a été validé, le passager suit son itinéraire initial annoncé par l'API en se déplaçant de son point de départ O_r à sa destination D_r. Dans un tel cas, le processus est ré-exécuté lorsque l'utilisateur arrive au prochain arrêt x_1. Autrement, si un des chemins en covoiturage a été validé, la route du passager doit être mise à jour à $O_r \rightsquigarrow x_i^\star \rightsquigarrow x_j^\star$, tel que le passager se déplace de O_r à $x_i^\star$ en utilisant les transports public, puis de $x_i^\star$ à $x_j^\star$ en covoiturage. Enfin, le processus se ré-exécute une fois que le passager atteint son lieu de dépose $x_j^\star$. Le nouvel itinéraire du conducteur devient $O_{k^*} \rightsquigarrow x_i^\star \rightsquigarrow x_j^\star \rightsquigarrow D_{k^*}$ (voir la figure 6.20).

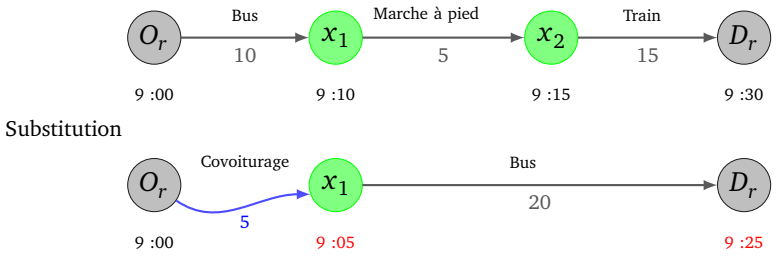


FIGURE 6.20 – Mise à jour de l'itinéraire du passager.

L'algorithme 30 détaille cette étape.

Algorithme 30 Calcul de plus courts chemins depuis les arrêts de dépose et détermination de la meilleure substitution

Entrée: Demande $O_r D_r$.
$t_a^r(D_r, p)$: le temps d'arrivée à la destination en transport public.
$\sigma^\star(x_i), t_a^\star(x_i)$ et $Driver\_stop(x_i), \forall x_i \in P$.
Sortie: Chemin multimodal combiné avec le covoiturage $P^\star$.
L'arrêt de prise en charge du passager $x_i^\star$.
L'arrêt de dépose du passager $x_j^\star$.
Mise à jour de l'itinéraire du conducteur $k^\star$.

1: **Initialisation** : $optimal\_arrival\_time \leftarrow t_a^r(D_r, p)$.
$k^\star \leftarrow -1$.
2: **Pour tout** $x_j \in P$ avec $\sigma^\star(x_j) > 0$ **Faire**
3: Calculer un plus court chemin multimodal classique $P = x_j, \ldots, x_{nbs}, D_r$ en utilisant l'API, partant de x_j à l'instant $t_a^\star(x_j)$, avec l'heure d'arrivée à la destination $arrival\_time\_at\_D_r$.
4: **Si** $arrival\_time\_at\_D_r < optimal\_arrival\_time$. **Alors**
5: $optimal\_arrival\_time \leftarrow arrival\_time\_at\_D_r$.
6: $k^\star \leftarrow Driver\_stop(x_j)$
7: Récupérer la route du conducteur $k^\star$ jusqu'à x_j, c'est à dire $O_k^\star \rightsquigarrow x_i \rightsquigarrow x_j$.
8: $x_i^\star \leftarrow x_i$
9: $x_j^\star \leftarrow x_j$
10: **Fin Si**
11: **Fin Pour**
12: **Si** $optimal\_arrival\_time < t_a^r(D_r, p)$ **Alors**
13: Mise à jour de l'itinéraire du conducteur $O_{k^\star} \rightsquigarrow x_i^\star \rightsquigarrow x_j^\star \rightsquigarrow D_{k^\star}$
14: Mise à jour de l'itinéraire du passager $P^\star$ à $O_r \rightsquigarrow x_i^\star \rightsquigarrow x_j^\star$
15: **Fin Si**

6.4.5 Accélération de l'approche

L'approche décrite ci dessous ne stocke aucune information sur les conducteurs autre que leur position actuelle. Cette dernière reste sensible aux longs trajets. Par exemple, un passager qui doit attendre quelques heures à un arrêt x_j pour effectuer une correspondance, l'espace de recherche autour des arrêts se situant après l'arrêt de correspondance x_j, prend en compte cette durée d'attente et devient rapidement très long (étape 3 de l'algorithme 28). Pour éviter cela, nous gardons la trajectoire de chaque conducteur depuis sa position actuelle jusqu'à sa destination finale. Basée sur ces informations, la durée $Max\_Duration(x_i)$ autour de l'arrêt x_i dépendra seulement de la longueur de la fenêtre de temps de prise en charge (c'est à dire $Max\_Duration(x_i) = d^{x_i}_{\max} - d^{x_i}_{\min}$). Par conséquent, l'admissibilité d'un chemin en covoiturage avec un conducteur k sera effectuée en considérant sa position actuelle ainsi que ses futures positions marquées par la recherche autour de l'arrêt de prise en charge x_i.
Pour ce faire, deux étapes de l'algorithme 28 seront modifiées :

- La durée maximale de l'espace de recherche autour d'un arrêt de transit x_i dans l'étape 3 sera réduite à $Max\_Duration(x_i) = (d^{x_i}_{\max} - d^{x_i}_{\min})$.
- L'étape 5 est étendue pour chaque position des trajectoires des conducteurs (pas seulement selon leur position actuelle), en vérifiant la synchronisation temporelle entre le conducteur et le passager. La première position marquée d'un conducteur k (position actuelle ou future) qui respecte la fenêtre de temps de prise en charge ne garantit pas un plus petit détour par rapport à ses autres positions autour de l'arrêt de transit x_i. Pour garantir un détour et une heure d'arrivée minimale pour le conducteur k autour de l'arrêt de transit x_i, une recherche selon $Max\_Duration(x_i)$ doit être totalement achevée. Enfin, les étapes 9, 10 et 11 sont uniquement effectuées pour les positions qui assurent une heure d'arrivée à x_i plus tôt que celles qui utilisent les transports publics.

Les trajectoires des conducteurs stockées peuvent être facilement mises à jour en fonction des conditions météorologiques ou du trafic.

6.4.6 Complexité de l'approche par étiquetage

La complexité de l'algorithme de Dijkstra dans un réseau routier et dans un réseau multimodal. L'étape 1 de l'algorithme 27 se détermine en $O(\mathscr{D}_{API})$. Les étapes 2, 3 et 4 de l'algorithme 27 peuvent être parallèlement déterminées en $O\big(\mathscr{D}_{road} \cdot (|P|-1)\big)$. L'algorithme 28 permet de déterminer la liste potentielle des conducteurs autour de chaque arrêt potentiel de prise en charge ainsi que les durées de détour restantes pour chaque arrêt potentiel de dépose. Ce dernier est déterminé en utilisant l'algorithme de Dijkstra inverse limité avec une complexité de $O\big(|P^+| \cdot \mathscr{D}_{road} + \sum_{i \in P^-} Nb\_Driver_i \cdot |P_i^{'}|\big)$, où $|P^+|$ est le nombre d'arrêts potentiels de prise en charge (c'est à dire les arrêts dont leur fenêtre de temps est non vide), Nb_Driver_i est le nombre de conducteurs marqués dans la *recherche avant* autour de l'arrêt i. Afin de sélectionner un meilleur conducteur pour chaque arrêt potentiel de dépose, l'algorithme 29 vérifie l'admissibilité de chaque conducteur potentiel autour des arrêts de dépose. Ce dernier est calculé en $O\big(|P^-| \cdot \mathscr{D}_{road} + \sum_{i \in P^+} \sum_{j \in DriverList_i} Driver\_routes^i_j\big)$, où $|P^-|$ est le nombre d'arrêts potentiels de dépose (c'est à dire les arrêts dont lesquels la réduction du temps d'arrivée en utilisant le covoiturage est positive), $DriverList_i$ est la liste des conducteurs potentiels détectés par la *recherche avant* autour de l'arrêt i et $Driver\_routes^i_j$ est la liste de routes potentielles du conducteur j vers l'arrêt de dépose i. Ainsi, parmi tous les arrêts de dépose admissibles, l'algorithme 30 sélectionne l'arrêt qui améliore le plus le temps d'arrivée du passager. Pour ce faire, nous calculons $|P^-|$ fois l'algorithme de Dijkstra dans le réseau multimodal depuis les arrêts de dépose vers la destination

du passager en prenant en compte la nouvelle heure d'arrivée. Enfin, la complexité de l'ensemble du processus est de $O\big(\mathscr{D}_{road} \cdot (|P| + |P^+| + |P^-| - 1) + \mathscr{D}_{API} \cdot (|P^+| + 1) + \sum_{i \in P^+} Nb\_Driver_i \cdot |P_i^{'}| + \sum_{i \in P^-} \sum_{j \in DriverList_i} Driver\_routes_j^i\big)$. Le pire des cas peut se produire lorsque chaque arrêt du chemin multimodal classique est potentiel pour une prise en charge et de dépose. Dans ce cas, la complexité est de $O\big(3 \cdot \mathscr{D}_{road} \cdot (|P| - 1) + \mathscr{D}_{API} \cdot |P| + \sum_{i \in P} |P_i^{'}| + \sum_{i \in P} \sum_{j \in DriverList_i} Driver\_routes_j^i\big)$.

6.5 Troisième approche : Approche par buckets

Dans les deux approches 6.3 et 6.4, la sollicitation pour un covoiturage se fait uniquement par le passager en ré-exécutant le processus à chaque arrêt de correspondance. Dans cette troisième approche, chaque nouvelle entrée dans le système (offre ou demande) active une recherche de covoiturage. Plus spécifiquement, l'arrivée d'un conducteur active une recherche de covoiturage sur l'ensemble des passagers existants et une arrivée d'un passager active le processus de recherche des conducteurs potentiels. Parmi tous les chemins admissibles en covoiturage, nous sélectionnons ceux qui améliorent l'heure d'arrivée du passager.
Nous décrivons le processus d'ajout d'une offre de covoiturage et d'une requête d'un passager. Nous utilisons deux types de "seaux" qui contiennent des informations sur les passagers et les conducteurs. Plus spécifiquement, le système maintient pour chaque lieu potentiel de prise en charge et de dépose, des informations sur les conducteurs et/ou les passagers qui sont potentiels pour un éventuel covoiturage. Afin de réduire l'espace de stockage, les lieux de prise en charge et de dépose pour les conducteurs sont limités à des lieux stratégiques, à savoir : les grandes stations, les arrêts de bus, etc.

Definition. 12 ***(Seau des conducteurs*** $Bucket_C(v)$ $v \in V$***)***
Le seau des conducteurs $Bucket_C(v)$ *associé à un arrêt* v *est l'ensemble de triplets où chaque triplet est composé de l'identifiant d'un conducteur* k*, l'heure d'arrivée à l'arrêt* v *et la dernière heure de départ de* v*, tel que le conducteur est capable de prendre en charge les passagers à* v *sans enfreindre la contrainte du temps de détour (6.2).*

$$Bucket_C(v) := \{(k, t_a^k(v,c), t_d^k(v,c)) | \textit{ contrainte } (6.2) \textit{ est satisfaite.}\} \tag{6.4}$$

Un conducteur k est inséré dans $Bucket_C(v)$ si et seulement si la durée de son trajet passant par l'arrêt v n'excède pas son temps de détour maximal, c'est à dire : $\delta(O_k, v) + \delta(v, D_k) \leq \lambda_k \delta(O_k, D_k)$. Les entrées d'un conducteur k sont ajoutées dans un bucket des conducteurs par ordre décroissant selon leur temps de départ au plus tard $t_d^k(v,c) = t_o^k + \lambda_k \delta(O_k, D_k) - \delta(v, D_k)$ (la dernière heure est la première dans le bucket $Bucket_C(v)$).

Definition. 13 ***(Seau des passagers*** $Bucket_P(v)$ $v \in V$***)***
Le seau des passagers $Bucket_P(v)$ *associé à un arrêt* v *est un ensemble de triplets où chaque triplet est composé de l'identifiant du passager* r*, l'heure d'arrivée à l'arrêt* v *et la dernière heure de départ de* v *en utilisant les transports publics.*

$$Bucket_P(v) := \{(r, t_a^r(v,p), t_d^r(v,p))\}. \tag{6.5}$$

En d'autres termes, un bucket $Bucket_P(v)$ contient la liste des passagers ainsi que leur fenêtre de temps de prise en charge. Les entrées de $Bucket_P(v)$ sont classées par ordre décroissant selon l'heure d'arrivée à l'arrêt concerné.

Nous maintenons également les chemins OD avec l'information des temps de passage pour chaque conducteur et passager jusqu'à ce que ces derniers quittent le système ou atteignent leur destination.

6.5.1 Ajout d'une offre d'un conducteur

Chaque nouvelle offre entrante dans le système génère de nouvelles opportunités de covoiturage. Soit k le nouveau conducteur souhaitant se déplacer à l'instant t_k de sa position actuelle O_k à sa destination finale D_K. Nous calculons l'intégration de cette nouvelle offre par une détermination de tous les arrêts possibles de prise en charge et de dépose. Pour cela, nous utilisons l'algorithme de Dijkstra $A^\star$ bidirectionnelle modifié (BA). Les deux files de priorité Q_1 (file de priorité pour l'espace de recherche avant) et Q_2 (file de priorité pour l'espace de recherche arrière) sont guidées. Ces dernières sont guidées par une fonction heuristique qui estime la durée restante pour atteindre la destination dans le cas d'une *recherche avant*, et l'autre durée restante pour atteindre le point de départ dans la *recherche arrière*. Ainsi, pour un nœud donné v, la fonction de coût utilisée dans la première file de priorité Q_1 est $\delta(O_k, v) + \hat{\delta}(v, D_k)$, tandis que dans la deuxième file de priorité cela correspond à $\hat{\delta}(O_k, v) + \delta(v, D_k)$ (voir la figure 6.21).

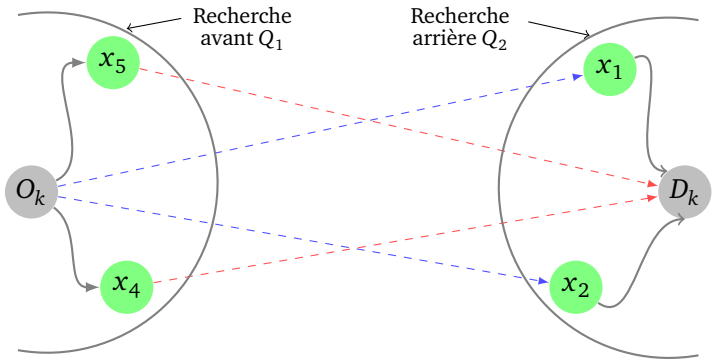


FIGURE 6.21 – Situation de l'algorithme BA avant la détermination de la durée $\delta(O_k, D_k)$. Les deux recherches sont guidées par une estimation des durées restantes. Les lignes en pointillé de couleur rouge correspondent aux durées estimées pour l'espace de recherche avant Q_1 et les lignes en pointillé de couleur bleu correspondent aux durées estimées pour l'espace de recherche arrière Q_2.
A ce niveau, uniquement le demi-disque à gauche autour de O_k est visité pour la recherche avant Q_1, et le demi-disque à droite autour de D_k pour la recherche arrière Q_2. Les lignes continues correspondent aux durées exactes déterminées séquentiellement après chaque marquage soit par la file Q_1, soit par la file Q_2.

La procédure de *relaxation d'arcs* appliquée reste similaire à l'algorithme de Dijkstra bidirectionnelle décrit dans le chapitre précédent. Plus spécifiquement, une fois que la durée $\delta(O_k, D_k)$ est déterminée, l'espace de recherche continue jusqu'à ce que tous les nœuds avec un coût provisoire plus petit que $\lambda_k \delta(O_k, D_k)$ soient marqués. A ce niveau, les nœuds restants à marquer se situent dans le demi-disque à droite autour de la cible O_k ou dans le demi-disque à gauche autour de D_k. La ligne rouge correspond à la durée $\delta(O_k, D_k)$. Nous gardons uniquement les nœuds qui satisfont la contrainte de détour (voir la figure 6.22).

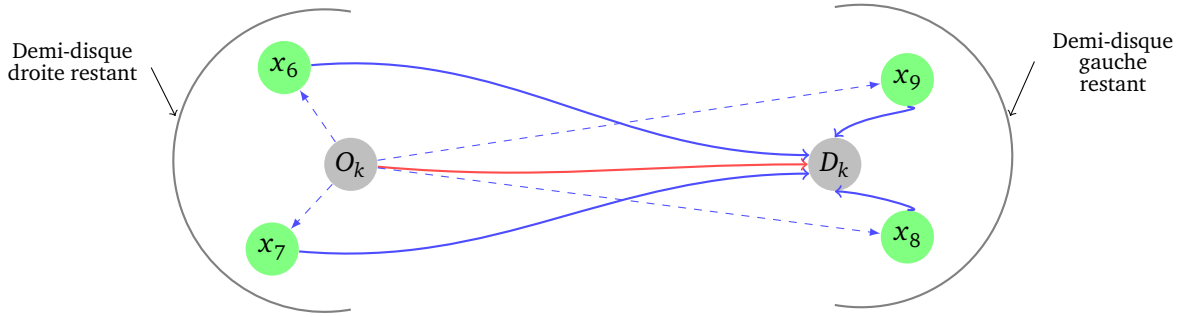


FIGURE 6.22 – Situation de l'algorithme (BA) après la détermination de la durée du trajet direct du conducteur $\delta(O_k, D_k)$.

Pour chaque nœud v marqué par les deux files de priorité Q_1 et Q_2, si la contrainte du *temps de détour* est respectée, alors les informations du conducteur sont insérées dans le *bucket* $Bucket_C(v)$ par ordre décroissant par rapport à la dernière heure de départ. Enfin, tous les noeuds marquées par les deux files de priorité et qui respectent la contrainte du *temps de détour*, leur buckets passagers $Bucket_P$ sera parcouru dans le but de sélectionner le meilleur passager avec qui le conducteur partagera son trajet. La manière de parcourir un bucket des passagers $Bucket_P(v)$ est décrite dans l'algorithme 31.

Algorithme 31 Parcours d'un bucket des passagers $Bucket_P(v)$

Entrée: Conducteur k, bucket $Bucket_P(v)$.
Sortie: Meilleur passager $r^\star$, meilleur arrêt de dépose $x_{j^\star}$.
$\sigma^\star$: plus grande économie de temps générée pour le passager $r^\star$ avec v comme arrêt de prise en charge.

1: **Initialisation** : $r^\star \leftarrow -1$, $\sigma^\star \leftarrow 0$, $x_{j^\star} \leftarrow -1$
2: **Pour tout** r **dans** $Bucket_P(v)$ **Faire**
3: **Pour tout** arrêt x_j **dans** P situé après l'arrêt v **Faire**
4: $\sigma \leftarrow t_a^r(x_j, p) - \left(\max\{t_a^k(v, c), t_a^r(v, p) + \tau(v)_{pc}\} + \delta(v, x_j)\right)$
5: **Si** $k \in Bucket_C(x_j)$ **et** (v, x_j) forme *une substitution admissible* pour le conducteur k et le passager r **et** $\sigma > \sigma^\star$ **Alors**
6: $x_{j^\star} \leftarrow x_j$, $r^\star \leftarrow r$, $\sigma^\star \leftarrow \sigma$
7: **Fin Si**
8: **Fin Pour**
9: **Fin Pour**

Ce dernier est appliqué pour chaque nœud marqué par les deux recherches Q_1 et Q_2.
Dans l'étape 5, avant de tester l'admissibilité de la substitution du trajet v et x_j entre le conducteur k et le passager r, nous vérifions tout d'abord si le conducteur k n'enfreint pas sa contrainte du temps de détour en passant par x_j (c'est à dire, si $k \in Bucket_C(x_j)$). Ainsi, le passager $r^\star$ ainsi que le lieu de prise en charge $x_{i^\star}$ et de dépose $x_{j^\star}$ correspondent à ceux qui génèrent le plus de réduction en temps d'arrivée vers leur destination. Les différentes étapes pour ajouter une offre de covoiturage sont définies dans l'algorithme 32.

Algorithme 32 Ajout d'une offre d'un conducteur k

Entrée: Conducteur k, coefficient de détour λ_k.
Sortie: Meilleur passager $r^\star$, meilleur arrêt de prise en charge $x_{i^\star}$, meilleur arrêt de dépose $x_{j^\star}$, mise à jour des buckets des conducteurs.

Initialisation : Q_1.*Insérer*(O_k) et Q_2.*Insérer*(D_k)
$List\_de\_noeuds\_potentiels \leftarrow \emptyset$, $Stop\_recherche \leftarrow$ Faux, $\delta(O_k, D_k) \leftarrow +\infty$
Tant que $Stop\_recherche$ =Faux **Faire**
  $Noeud\_courant \leftarrow \min(Q_1.\min(), Q_2.\min())$
  Marquer $Noeud\_courant$ comme visité en le retirant de la file de priorité appropriée
  Si $Noeud\_courant$ est marqué par les deux files de priorité Q_1 et Q_2 **Alors**
    $List\_de\_noeuds\_potentiels \leftarrow List\_de\_noeuds\_potentiels \cup \{Noeud\_courant\}$
    Si $\delta(O_k, Noeud\_courant) + \delta(Noeud\_courant, D_k) < \delta(O_k, D_k)$ **Alors**
      Mise à jour de $\delta(O_k, D_k) \leftarrow \delta(O_k, Noeud\_courant) + \delta(Noeud\_courant, D_k)$
    Fin Si
    Si $\min\big(\delta(O_k, Noeud\_courant), \delta(Noeud\_courant, D_k)\big) > \lambda_k \delta(O_k, D_k)$ **Alors**
      $Stop\_recherche \leftarrow$ Vrai
    Fin Si
  Fin Si
  Mise à jour des coûts provisoires des noeuds voisins du $Noeud\_courant$ en les insérant dans la file appropriée selon la fonction de coût $\big(\delta(O_k, Noeud\_courant) + \hat{\delta}(Noeud\_courant, D_k)$ pour Q_1 et $\hat{\delta}(O_k, Noeud\_courant) + \delta(Noeud\_courant, D_k)$ pour $Q_2\big)$.
Fin Tant que
Pour tout v dans $List\_de\_neouds\_potentiels$ **Faire**
  Si $\delta(O_k, v) + \delta(v, D_k) \leq \lambda_k \delta(O_k, D_k)$ **Alors**
    $t_a^k(v, c) \leftarrow t_o^k + \delta(O_k, v)$
    $t_d^k(v, c) \leftarrow t_o^k + \lambda_k \delta(O_k, D_k) - \delta(v, D_k)$
    $Bucket_C(v) \leftarrow Bucket_C(v) \cup \{(k, t_a^k(v, c), t_d^k(v, c))\}$
    {l'opération d'insértion est dans l'ordre décroissant en fonction de $t_d^k(v, c)$}
    Parcourir le bucket des passagers $Bucket_P(v)$ en utilisant l'**algorithme 31**
    Mise à jour du meilleur passager $r^\star$, meilleur arrêt de prise en charge $x_{i^\star}$ et meilleur arrêt de dépose $x_{j^\star}$
  Fin Si
Fin Pour

La potentialité des durées économisées sur chaque arrêt de correspondance peut être validée en calculant le plus court chemin en transport public à partir de l'arrêt associé en prenant en compte la nouvelle heure d'arrivée vers la destination du passager.

6.5.2 Ajout d'une requête d'un passager

Le principe d'ajout d'une requête d'un passager reste le même que celui des deux approches décrites dans la section 6.3 et 6.4. Les deux différences sont : (i) certaines informations des conducteurs sont déjà stockées dans les buckets pour accélérer le calcul lors de l'arrivée d'une requête (ii) dans le cas où aucun covoiturage n'a été sélectionné, les informations du passager courant sont temporairement stockées dans les buckets des passagers pour d'éventuelles arrivées des conducteurs. Notons par $O_r D_r$ l'origine-destination du passager débutant son trajet à t_r. Les différentes étapes sont décrites ci-dessous :

- Étape d'initialisation

La première étape est similaire à celle décrite dans l'approche précédente 6.3. En se basant sur un itinéraire multimodal classique P renvoyé par l'API. Cette étape détermine les possibilités en covoiturage entre les différents arrêts de correspondance de P qui diminuent l'heure d'arrivée du passager à sa destination ainsi que les fenêtres de prise en charge sur chaque arrêt (voir les figures 6.11, 6.12 et 6.13).

- Conducteurs admissibles

Le but de cette étape est de déterminer si elle existe, la meilleure substitution en covoiturage sur le trajet multimodal classique P. La première phrase consiste à déterminer l'ensemble des conducteurs capables de prendre en charge le passager sur les arrêts de correspondance renvoyés par l'API en fonction de leurs fenêtres de temps. Cette phase comprend également la détermination des durées des conducteurs depuis leur position actuelle vers chaque arrêt potentiel de prise en charge. Ces durées sont calculées en utilisant l'algorithme de Dijkstra inverse depuis chaque arrêt potentiel de prise en charge. Ces dernières n'ont pas été stockées dans les buckets car les positions des conducteurs en circulation changent à tout moment. En revanche, les informations contenues dans les buckets des conducteurs $Bucket_C(x_i)$ permettent d'éviter de re-calculer les plus courts chemins depuis x_i vers les destinations des conducteurs potentiels (car les destinations des conducteurs sont fixes). A chaque étape lorsqu'un conducteur k est marqué pendant l'opération de recherche autour de x_i, nous vérifions tout d'abord si ce conducteur se trouve dans le bucket $Bucket_C(x_i)$. Si ce n'est pas le cas, le détour maximal toléré par le conducteur en passant par x_i n'est plus respecté. Sinon, d'autres tests seront effectués. Nous parcourons la liste des chemins potentiels ayant x_i comme lieu de prise en charge (c'est à dire $DriverList(x_i)$) en testant si le conducteur k est dans le bucket $Bucket_C(x_j)$. Si cela est vérifié, nous testons donc l'*admissibilité du chemin* (x_i,x_j) en mettant à jour si nécessaire, le meilleur gain généré $(x_{i^\star},x_{j^\star})$ ainsi que le conducteur associé $k^\star$. Enfin, pour chaque arrêt potentiel de dépose x_j, nous avons associé le meilleur conducteur qui génère le meilleur gain en temps d'arrivée vers l'arrêt concerné.

Algorithme 33 Meilleur conducteur associé à chaque arrêt potentiel de dépose.

Entrée: $DriverList(x_i)$: conducteurs potentiels ayant x_i comme lieu de prise en charge, $\forall x_i \in P$, Demande r.
Sortie: *Meilleur_conducteur*(x_j) : Meilleur conducteur associé à chaque arrêt potentiel de dépose x_j.
Meilleur_temps_arrive(x_j) : Meilleur temps d'arrivée associé à chaque arrêt potentiel de dépose x_j.

Pour tout x_i dans P, avec une fenêtre de temps non vide **Faire**
  Initialisation : $k^\star \leftarrow -1$
  Pour tout noeud O_k marqué par la recherche autour x_i correspondant à une position actuelle d'un conducteur **et** $k \in Bucket_C(x_i)$ **Faire**
    Pour tout (x_i, x_j) dans la liste $DriverList(x_i)$ **Faire**
      Si $k \in Bucket_C(x_j)$ **et** (x_i, x_j) est un chemin *admissible de substitution* pour le conducteur k et le passager r **Alors**
        Si $\sigma^\star \geq t_a^r(x_j, p) - \left(\max\{t_a^r(x_i, p) + \tau(x_i)_{pc}, t_a^k(x_i, c)\} + \delta(x_i, x_j)\right)$ **Alors**
          $\sigma^\star \leftarrow t_a^r(x_j, p) - \left(\max\{t_a^r(x_i, p) + \tau(x_i)_{pc}, t_a^k(x_i, c)\} + \delta(x_i, x_j)\right)$
          Meilleur_conducteur$(x_j) \leftarrow k^\star$
          Meilleur_temps_arrive$(x_j) \leftarrow \left(\max\{t_a^r(x_i, p) + \tau(x_i)_{pc}, t_a^k(x_i, c)\} + \delta(x_i, x_j)\right)$
        Fin Si
      Fin Si
    Fin Pour
  Fin Pour
Fin Pour

- Meilleure substitution et mise à jour de l'itinéraire du passager

Cette étape est similaire à celle décrite dans l'approche 6.4.4. La seule différence se trouve dans le cas où aucun covoiturage n'a été sélectionné. Dans ce cas, les buckets des passagers pour chaque arrêt potentiel de prise en charge est mis à jour selon la définition 13.

6.5.3 Mise à jour des buckets

Chaque fois qu'un nœud v est visité, le bucket est mis à jour de sorte que toutes les informations obsolètes soient supprimées. Comme les buckets sont triés par ordre décroissant par rapport à l'heure de départ au plus tard, la suppression de ces éléments se fait en temps linéaire. Dans notre application, les buckets sont réinitialisés toutes les 24 heures.

6.6 Expérimentations

Dans cette section, nous présentons les résultats numériques de nos trois approches proposées. Les algorithmes ont été codés en C# avec une machine équipée d'un processeur Intel (R) Core (TM) i7-3520M CPU 2,90 GHz et 8 Go de mémoire RAM. Un tas binaire a été utilisé comme structure de données de la file de priorité de l'algorithme de Dijkstra.
Dans nos expérimentations, nous avons utilisé les mêmes données ainsi que le même réseau routier décrit dans la section 3.5.2 avec le même filtrage sur l'ensemble des offres et des demandes de covoiturage. Le nombre de demande s'élève à 538, tandis que le nombre d'offre est de 537 offres.
Pour chaque demande, nous récupérons l'itinéraire multimodal classique en utilisant l'API de Canal TP [28]. Canal TP est une filiale numérique du groupe Keolis [29] qui est leader en France du calcul d'itinéraire et de l'information en temps réel. Cette dernière propose des solutions modulaires pour

28. http ://www.canaltp.fr
29. http ://www.keolis.com/

valoriser l'offre de transport et simplifier chaque déplacement. Les solutions d'information-voyageur de Canal TP s'appuient sur la technologie open source Navitia [30], fiable et reconnue depuis plus de dix ans. Navitia est un service web qui offre de nombreuses fonctions dont la recherche isochrone, la recherche d'horaires, la recherche d'arrêts à proximité d'un point et le calcul d'itinéraires multimodaux classiques. Chacune de ces fonctions tire partie de l'information en temps réel.
Dans ces expérimentations, nous visons un double objectif :

(i) Le premier est de mesurer l'efficacité de nos trois approches par rapport au système multimodale classique retourné par l'API en termes de nombre de trajets améliorés et de gains générés en temps d'arrivée.

(ii) Le second est de mesurer le temps de traitement des requêtes pour les trois différentes approches.

Notre scénario est le suivant : nous parcourons chaque demande d'un passager, et pour chaque demande, nous considérons toutes les offres (c'est à dire les 537 offres).
Nous évaluons d'abord le nombre de trajets améliorés en utilisant nos différentes approches par rapport au transport multimodal classique déterminé par l'API. Pour cela, nous avons divisé les instances en trois groupes :

- *No ride* : le pourcentage des demandes satisfaites uniquement par le transport multimodal classique (API).
- *Ride only* : le pourcentage des demandes satisfaites uniquement par le covoiturage classique.
- *Both* : le pourcentage des demandes satisfaites en combinant les transports publics et le covoiturage.

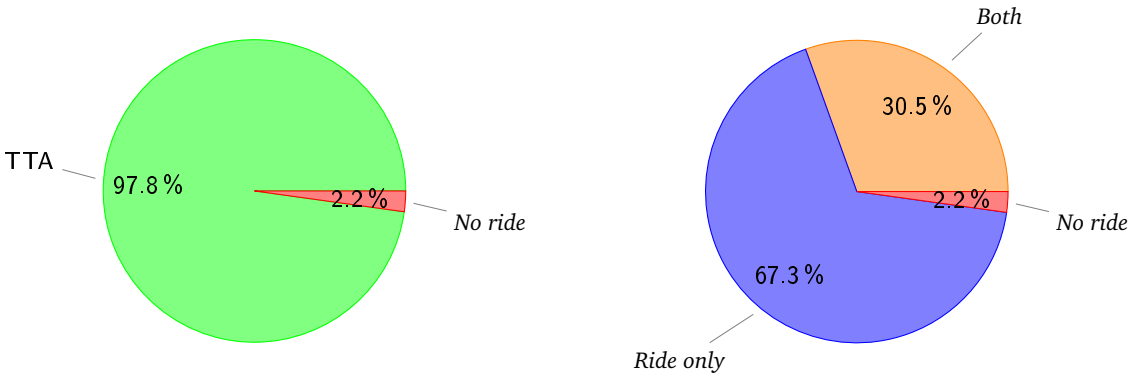


FIGURE 6.23 – Pourcentage du nombre de trajets pour chaque groupe, TTA (nombre Total de Trajets Améliorés), *No ride*, *Ride only* et *Both*.

La figure 6.23 indique que 97,8% des trajets se sont améliorés en termes d'heure d'arrivée comparé aux trajets retournés par l'API. Parmi eux, 67,3% sont composés uniquement d'un seul covoiturage et 30,5% combinent le covoiturage et le transport multimodal classique.

La figure 6.24 représente la fréquence du nombre de correspondance retournée par l'API qui varie de 2 à 8. Un trajet avec deux arrêts (origine et destination) peut se produire lorsque le trajet initial est composé uniquement d'un mode de déplacement (transport public ou marche à pied). La majeure partie des déplacements est composée de quatre correspondances. En effet, l'origine et la destination sont considérées comme des correspondances. Il est à noter qu'un utilisateur peut fixer un nombre maximal de correspondance retourné par l'API. Dans notre approche, ce nombre n'est pas fixé au préalable puisque notre objectif est d'amener le passager le plus tôt possible à sa destination

30. http ://www.navitia.io/

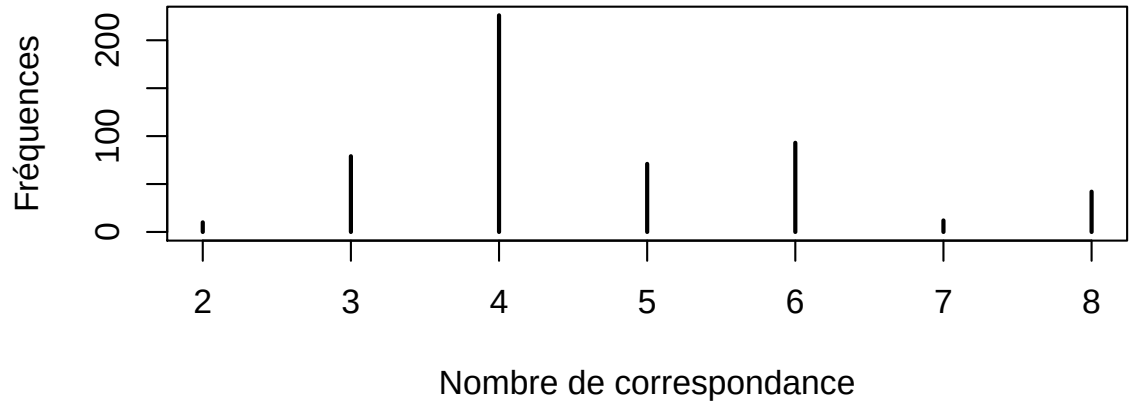


FIGURE 6.24 – Histogramme du nombre de correspondance.

finale.

Dans cette seconde partie, nous présentons le temps d'exécution des requêtes et nous comparons également le gain économisé en heure d'arrivée que nos approches génèrent par rapport aux heures d'arrivées du transport multimodal classique. Les trois approches génèrent les mêmes économies en heure d'arrivée sur l'ensemble des instances vu que les trois approches sont basées sur l'itinéraire multimodale classique API. Le gain est divisé en deux groupes : le groupe composé uniquement de covoiturage ($Gain_{Ride\ only}$) et le groupe ($Gain_{Both}$) combinant les deux services. A cet effet, nous notons par :

- $Gain$: le temps économisé en heure d'arrivée par rapport au transport multimodal classique.
- $Gain_{Ride\ only}$: le temps économisé en heure d'arrivée pour le groupe *Ride only* par rapport au transport multimodal classique.
- $Gain_{Both}$: le temps économisé pour le groupe *Both* par rapport au transport multimodal classique.
- $Time_i$: la durée de traitement d'une requête en *seconde* pour la approche i, $i = \{$ *directe, étiquetage, buckets*$\}$.

Médiane						
Gain			Durée des trajets	Durée de traitement		
$Gain$ (min)	$Gain_{Ride\ only}$ (min)	$Gain_{Both}$ (min)	(min)	$Time_{directe}$ (s)	$Time_{étiquetage}$ (s)	$Time_{buckets}$ (s)
52	55	24.43	84	52.85	6.75	3.83

TABLE 6.2 – Performance des trois approches.

Les valeurs dans le tableau 6.2 correspondent aux médianes de l'ensemble des instances. La colonne "Durée des trajets" représente la valeur médiane des durées des trajets initiaux retournée par l'API. Cette durée comprend également les temps d'attentes effectués aux différentes correspondances. Comme nous pouvons le constater, les gains générés en heure d'arrivée sont très importants. Ces derniers atteignent approximativement une heure dans le cas où la fréquence des transports publics est faible. Le gain généré dans le groupe *Ride only* est plus élevé car les passagers sont directement pris en charge depuis leur origine et déposés à leur destination. La durée médiane de traitement d'une requête est raisonnable pour une application en temps réel. *L'approche par étiquetage* et

l'approche par buckets sont plus rapide comparées à *l'approche directe*. Ceci est dû aux limitations effectuées sur les recherches des conducteurs potentiels en définissant des rayons de recherche autour des arrêts de correspondance.
Pour une meilleure visualisation, nous présentons dans la figure 6.25 le temps de traitement d'une requête en fonction du nombre de conducteurs potentiels pour *l'approche directe* et *l'approche par étiquetage*.

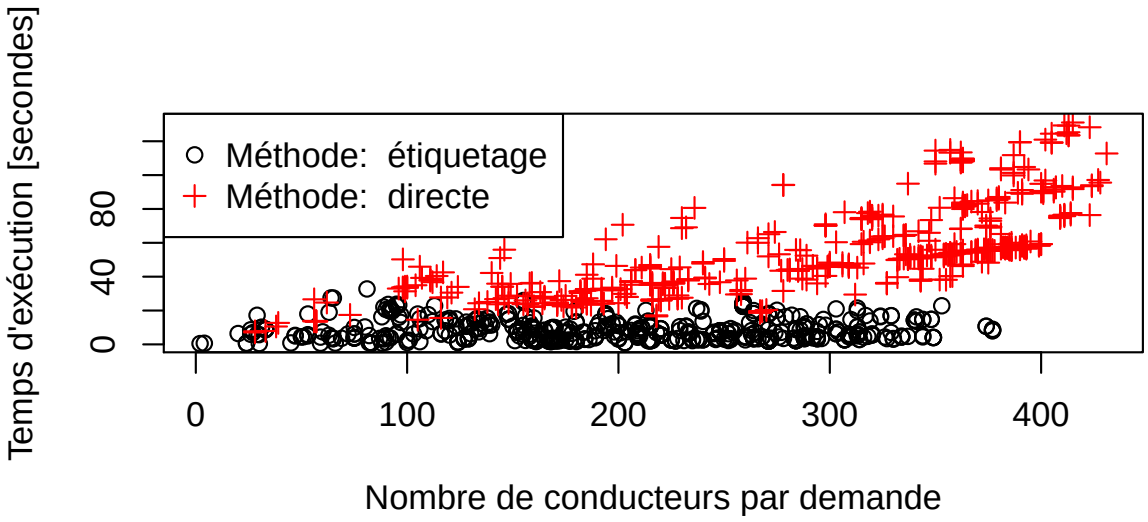


FIGURE 6.25 – Temps de traitement des requêtes selon le nombre de conducteurs potentiels. Les cercles de couleur noire correspondent à l'*approche par étiquetage*, tandis que les '+' de couleur rouge font référence à l'*approche directe*.

D'après la figure 6.25, la différence des durées maximales de traitement d'une requête pour les deux approches est très significative. Plus exactement, la durée maximale dans *l'approche directe* atteint 2 minutes, alors que dans *l'approche par étiquetage* elle ne dépasse pas 35 secondes. Une autre information qui peut être déduite à partir de cette figure concerne le nombre de conducteurs potentiels détecté. Comme on peut le constater sur la figure de projection des offres (figure 3.12), les offres sont proches les unes des autres, d'où le fait que le nombre de conducteurs potentiels est élevé. Ces conducteurs correspondent aux conducteurs qui ont été sélectionnés par l'étape d'estimation de proximité (section 6.3.2). Pour *l'approche directe*, le nombre de conducteurs potentiels détecté atteint 400 conducteurs par instance. En revanche, pour *l'approche par étiquetage* ce nombre ne dépasse pas 390 conducteurs dans toutes les instances. Ceci confirme également l'efficacité des recherches délimitées effectuées autour des correspondance dans la deuxième et la troisième approche.
La médiane de la durée de traitement d'une requête dans *l'approche par buckets* est plus petite que celle de l'*approche par étiquetage* car les durées depuis les arrêts de dépose vers les destinations des conducteurs potentiels ne sont pas calculées au moment d'une arrivée d'une requête. En effet, elles sont déjà stockées dans les buckets et la détermination de ces durées nécessitent seulement le parcours des buckets des arrêts de dépose.
La figure 6.26 représente le temps de traitement des requêtes en fonction du nombre de correspondance.

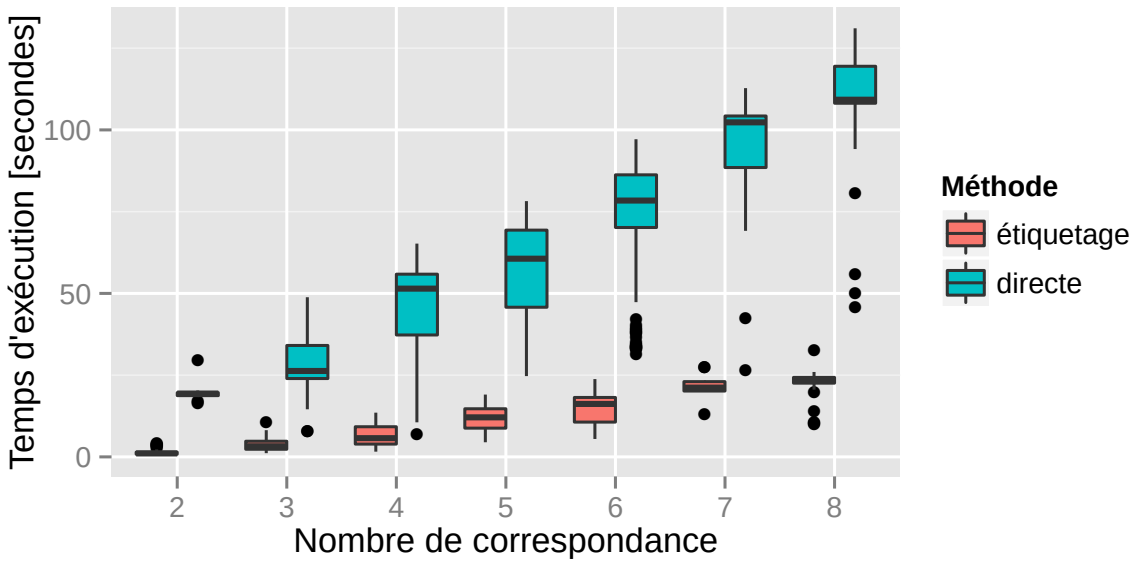


FIGURE 6.26 – Le temps de traitement des requêtes selon le nombre de correspondance (*approche directe et approche par étiquetage*).

Enfin, les deux figures 6.27 et 6.28 présentent une vue globale sur la variation de la durée de traitement des requêtes en fonction du nombre de correspondance ainsi que le nombre de conducteurs potentiels pour les deux approches, *l'approche directe* et *l'approche par étiquetage*.

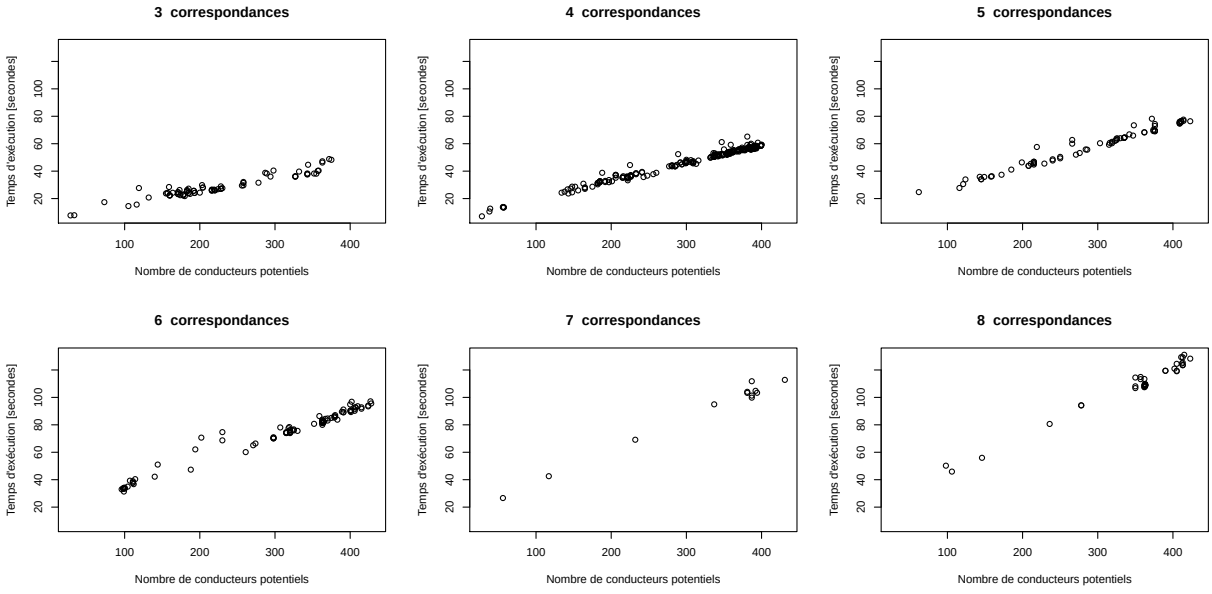


FIGURE 6.27 – Temps de traitement des requêtes en fonction du nombre de correspondance et du nombre de conducteurs potentiels (*approche directe*).

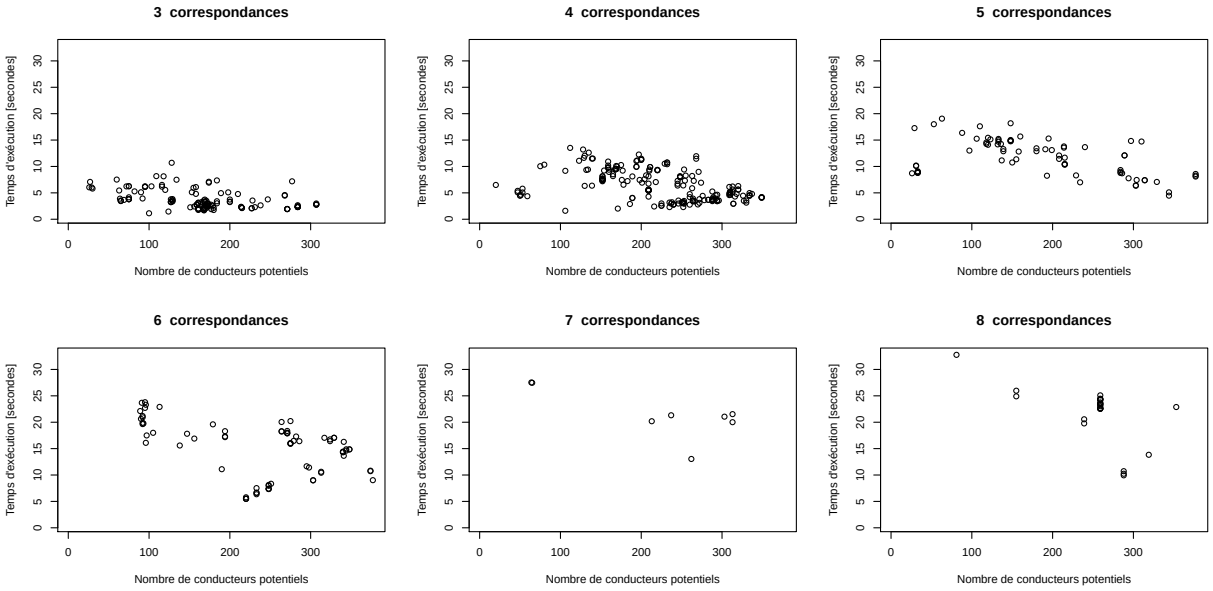


FIGURE 6.28 – Temps de traitement des requêtes en fonction du nombre de correspondance et du nombre de conducteurs potentiels (*approche par étiquetage*).

On remarque que le temps de traitement d'une requête augmente linéairement avec le nombre de conducteurs potentiels et le nombre de correspondances pour *l'approche directe*. Cependant, pour *l'approche par étiquetage*, la durée de traitement dépend uniquement du nombre de correspondance et non du nombre et des positions des conducteurs potentiels. Ces dernières influencent le temps d'exécution des algorithmes de Dijkstra utilisé dans *l'approche directe*. Pour résumer, *l'approche par étiquetage* est plus performante que *l'approche directe* en termes de temps de calcul. En revanche, l'utilisation de ces deux dernières approches (*l'approche par étiquetage* et *l'approche par buckets*) nécessite une carte géographique pour pouvoir utiliser la procédure de marquage. Autrement dit, la présence d'une carte permet l'étiquetage et le suivi en temps réel des positions des conducteurs. Basée sur ces étiquettes, la recherche limitée autour de chaque arrêt de correspondance est possible, ce qui n'est pas le cas dans *l'approche directe*.

6.7 Conclusion du chapitre

Dans ce chapitre, nous avons proposé système de transport multimodal innovant. Il s'agit d'une extension des systèmes multimodaux classiques en intégrant le service de covoiturage dans les modes de transport d'un usager. Notre modèle prend en considération d'importantes contraintes comme le temps d'attente du conducteur avant la prise en charge d'un passager et le temps de détour maximal du conducteur.
Les résultats expérimentaux montrent l'efficacité de nos approches comparées au système de transport multimodal classique en termes de gains générés sur les heures d'arrivées des usagers. Notre approche réduit également le détour du conducteur en lui offrant la possibilité d'utiliser des emplacements intermédiaires de prise en charge et de dépose. Ces lieux de prise en charge et de dépose sont limités aux arrêts de correspondance d'un itinéraire multimodal classique dans le but est de réduire d'une part, la combinatoire sur le nombre d'arrêts possibles, d'autre part les efforts effectués

par le passager sur la planification de leurs itinéraires. De plus, se baser sur l'itinéraire d'un système multimodal classique permet de garantir aux passagers un itinéraire fiable dans le cas d'une annulation ou d'un désistement en temps réel d'un covoiturage.
L'intérêt principal de ce modèle est de rendre le service de covoiturage et le transport en commun plus flexibles et plus efficaces. Le service de covoiturage peut être considéré comme un complément aux transports publics en couvrant les zones rurales difficiles à desservir par les transports publics. Ceci est la principale raison qui incite les organismes de transport public à utiliser le covoiturage pour compléter leur service. Malgré l'importance relative d'intégrer le covoiturage dans les services de transports publics, aucune étude permettant de faire face à ce problème dans son ensemble est effectuée. C'est à dire, la prise en compte du contexte dynamique et des différents conducteurs en temps réel. Une des raisons principales est la difficulté de définir et de combiner en temps réel les deux services : le covoiturage et les transports publics.

7

Conclusions et perspectives

Conclusions

Dans cette thèse nous avons développé plusieurs modèles de déplacement des usagers efficaces qui permettent de renforcer la mobilité partagée en répondant à la fois aux exigences des utilisateurs et de l'environnement. Comme nous l'avons constaté sur l'importance des véhicules des particuliers dans les modes de transport, ces derniers sont incontournables et le nombre de ces véhicules ne cesse d'augmenter. Ainsi, des services de mobilité partagée sont nécessaires en les combinant avec les différents modes de transport.
Pour ce faire, nous avons proposé différents services : le covoiturage, taxi-partagé ainsi que le routage multimodal incluant le service de covoiturage dynamique. Ces derniers répondent aux besoins de services de routage innovants et à une utilisation plus efficace de l'infrastructure des transports disponibles. Cette panoplie de services est un élément important pour le développement durable. En effet, le service de covoiturage peut pallier l'absence de transports en commun dans de nombreuses zones péri-urbaines ou semi-rurales. Cela permet également de limiter les embouteillages quotidiens ou l'isolement géographique.
Tout d'abord, nous avons commencé par développer des modèles généralisés pour le problème de covoiturage en étendant les systèmes classiques existants. Ces extensions ont permis de rendre le système de covoiturage plus flexible et attractif en développant deux modèles : un modèle de covoiturage dynamique en intégrant la notion de lieux de rencontre et de séparation (chapitre 3), et le modèle de covoiturage aller-retour avec stations relais (chapitre 4).
Par la suite, nous avons développé un nouvel algorithme de résolution pour le problème Dynamique Dial-a-Ride appliqué à une centrale de mobilité de taxis en prenant en compte des contraintes réelles. Il s'agit du temps de détour des passagers, du coût de trajet, du temps d'attente maximal avant la prise en charge, des lieux intermédiaires de prise en charge et de dépose pour les passagers. Nous avons également étendu ce modèle en considérant une flotte mixte composée de taxis et de covoiturage. Après avoir développé ces briques élémentaires, nous avons développé dans le chapitre 7 un modèle de transport multimodal innovant qui permet de combiner en temps réel le covoiturage dynamique dans les différents modes de transport.

Malgré les avantages que la mobilité partagée offre aux usagers ainsi qu'à l'environnement, cette dernière rencontre plusieurs freins d'utilisation : la sécurité, l'indépendance et la liberté procurée par la voiture individuelle. Parmi ces derniers, on peut retrouver :

– « Voyager avec des inconnus » qui est un classique des études sur le covoiturage. Des sites de covoiturage permettent de connaître les utilisateurs grâce à des fonctionnalités de réseaux

sociaux. C'est le cas par exemple de Covivo [31] ou BlaBlacar [32]. Plus précisément, ces outils permettent d'obtenir le profil des utilisateurs enregistrés, de voir leur préférences (accepte les animaux ou non, aime discuter ou non, etc), et d'avoir comme sur eBay une évaluation par les membres qui ont déjà covoituré avec le conducteur (ou le passager).
- «Monter dans une voiture dont on ne sait rien», la crainte est légitime, mais encore une fois la technologie du covoiturage dynamique permet d'authentifier les usagers. De plus, une étude de Muriel Dufresne, ingénieur transport à l'Agence De l'Environnement et de la Maîtrise de l'Energie [33] (ADEME), a montré que le covoiturage était moins accidentogène : on fait plus attention quand il y a du monde dans sa voiture.
- «Trop compliqué d'organiser des parcours et des emplois du temps», si c'était le cas avec le covoiturage classique, le covoiturage dynamique synchronise en temps réel les utilisateurs (grâce à un système « push » plus innovant). Certains permettent à leurs applications d'être enregistrées sur les smartphones ou sur les GPS de voiture.

Perspectives

Une bonne partie des problèmes considérés dans cette thèse sont des sujets d'actualité en recherche et ouvrent un grand nombre de perspectives.
Dans ce qui suit, nous présentons quelques perspectives de recherche qui peuvent être envisagées à court et à moyen terme :

- Premièrement, en extension des travaux des chapitres 4,5, 6 et 7, on peut envisager de prendre en considération plusieurs critères afin de répondre au mieux aux préférences des utilisateurs comme le coût monétaire total pour le passager, le détour du trajet origine-destination pour le conducteur, le nombre de correspondance, etc. Il existe deux façons pour intégrer ces fonctionnalités, la première est de considérer une seule fonction objectif définie comme la somme pondérée des différents critères. La deuxième façon consiste à optimiser tous les critères simultanément. Dans ce dernier, il n'existera pas qu'une seule solution mais un ensemble de solutions de compromis pour lesquelles il n'existe pas de meilleure solution sur l'ensemble des objectifs. La dominance de *Pareto* permet de définir les relations entre les solutions d'un problème multi-objectif. Il serait plus judicieux dans ce cas de considérer une approche interactive dans laquelle les préférences de l'utilisateur vont s'affiner au fur et à mesure pendant le processus de résolution.

- Deuxièmement, les deux approches de résolution proposées dans le chapitre 6 (problème dynamique de Dial-a-Ride avec fenêtre de temps) peuvent être étendues pour soulever la contrainte sur la fixation de l'ordre relatif des arrêts pendant l'insertion d'une nouvelle demande. Autrement dit, le nouveau lieu de prise en charge (resp. de dépose) ne sera pas forcément inséré entre deux arrêts successifs. En se basant sur les coûts de trajets des passagers ainsi que leur heure maximale d'arrivée, le nombre de combinaison de routes admissibles sera moins important. Cette extension sera significative dans le cas où les capacités des véhicules sont plus élevées, par exemple dans le cas de la flotte qui est composée de mini-bus.
  Une amélioration possible concerne le retour des décisions prises sur l'affectation des clients. Dans notre modèle, une fois que les requêtes sont admises, leur affectation n'est jamais modi-

31. www.covivo.fr
32. www.blablacar.fr
33. http ://www.ademe.fr/

fiée, même si cela pourrait engendrer une meilleure solution pour ces clients et une réduction sur le nombre de requêtes rejetées. En effet, dans certaines situations, il suffirait de modifier légèrement les trajets déjà planifiés pour être en mesure d'éviter des rejets d'éventuelles nouvelles requêtes. Pour y remédier, on peut imaginer une méthode qui interviendra uniquement lorsqu'une requête est rejetée de par des contraintes de capacité, de temps ou de coût du trajet. Les requêtes qui peuvent être réaffectées en s'assurant également la satisfaction de toutes les autres contraintes du modèle sont seulement celles qui sont planifiées mais qui n'ont pas encore été prise en charge.

– Le troisième perspective concerne le modèle développé dans le chapitre 7 sur le routage multimodal. En effet, ce modèle peut être étendu pour soulever la contrainte sur le nombre de passagers dans un véhicule privé. Jusqu'à présent, ce dernier est limité à un passager par véhicule. On peut s'inspirer de la procédure d'insertion développée dans le chapitre 6 afin de soulever la contrainte du nombre de passagers par véhicule. L'idée est de filtrer les arrêts de correspondances constituant le trajet multimodal traditionnel dans le but de sélectionner les arrêts potentiels de prise en charge et de dépose. Ensuite, chaque arrêt potentiel de prise en charge (resp. dépose) sera considéré comme un lieu de départ (resp. lieu d'arrivé) avec la fenêtre de temps adéquate. Une fois que ces derniers sont déterminés, nous procédons à la procédure d'insertion présentée dans le chapitre 6 avec quelques ajustements.

– Une autre perspective concerne l'interrogation de l'aspect monétaire sur un trajet mutimodal. En effet, les différentes approches proposées dans le chapitre 7 ne prennent pas en compte les coûts des trajets. Ainsi, ces approches peuvent être améliorées en définissant un coût global aux usagers comprenant à la fois les coûts des transports publics et le covoiturage.
La prise en compte des deux coûts doit être fait de sorte que le covoiturage puisse intervenir à la fois dans un milieu urbain et inter-urbain. En effet, un utilisateur souhaitant minimiser ses coûts de voyage et qui possède un abonnement mensuel pour les transports publics ne peut être attiré par un covoiturage que si ce dernier le fait arriver plus tôt ou dans le cas où les services de transport publics ne lui permettent pas d'atteindre sa destination finale. Pour éviter ceci, un équilibre entre ces différents facteurs peut être défini en appliquant par exemple les modèles de la *théorie des jeux*.

– Un dernier point important est de considérer les véhicules électriques dans le système de covoiturage et de taxis. Moins onéreux à l'usage que les véhicules thermiques, ce changement se fait ressentir par le déploiement d'infrastructures de recharge sur plusieurs territoires. A la différence des véhicules thermiques, les véhicules électriques sont limités par l'autonomie de leur batterie. Ainsi, l'intégration de ces derniers nécessiterait un autre modèle d'optimisation adapté à leur écosystème.

Bibliographie

[Agatz et al., 2011] Agatz, N. A., Erera, A. L., Savelsbergh, M. W., and Wang, X. (2011). Dynamic ride-sharing : A simulation study in metro atlanta. *Transportation Research Part B : Methodological*, 45(9) :1450–1464.

[Aissat and Oulamara, 2014a] Aissat, K. and Oulamara, A. (2014a). Dynamic ridesharing with intermediate locations. In *In Symposium Series on Computational Intelligence (SSCI)*.

[Aissat and Oulamara, 2014b] Aissat, K. and Oulamara, A. (2014b). A priori approach of real-time ridesharing problem with intermediate meeting locations. *Journal of Artificial Intelligence and Soft Computing Research*, 4(4) :287–299.

[Aissat and Oulamara, 2015a] Aissat, K. and Oulamara, A. (2015a). A posteriori approach of real-time ridesharing problem with intermediate locations. In *ICORES 2015 - Proceedings of the International Conference on Operations Research and Enterprise Systems, Lisbon, Portugal, 10-12 January, 2015.*, pages 63–74.

[Aissat and Oulamara, 2015b] Aissat, K. and Oulamara, A. (2015b). The round-trip ridesharing problem with relay stations. In *Computational Logistics - 6th International Conference, ICCL 2015 Delft, The Netherlands, September 23-25, 2015. Proceedings*, pages 16–30.

[Aissat and Varone, 2015a] Aissat, K. and Varone, S. (2015a). Carpooling as complement to multi-modal transportation. In *Enterprise Information Systems - 17th International Conference, ICEIS 2015, Barcelona, Spain, April 27-30, 2015, Revised Selected Papers*, pages 236–255.

[Aissat and Varone, 2015b] Aissat, K. and Varone, S. (2015b). Real-time ride-sharing substitution service in multi-modal public transport using buckets. In *Modelling, Computation and Optimization in Information Systems and Management Sciences - Proceedings of the 3rd International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences - MCO 2015, Metz, France, May 11-13, 2015, Part II*, pages 425–436.

[Antsfeld and Walsh, 2012] Antsfeld, L. and Walsh, T. (2012). Finding optimal paths in multi-modal public transportation networks using hub nodes and transit algorithm. *ARTIFICIAL INTELLIGENCE AND LOGISTICS*, page 7.

[Arz et al., 2013] Arz, J., Luxen, D., and Sanders, P. (2013). Transit node routing reconsidered. In *Experimental Algorithms*, pages 55–66. Springer.

[Baldacci et al., 2004] Baldacci, R., Maniezzo, V., and Mingozzi, A. (2004). An exact method for the car pooling problem based on lagrangean column generation. *Operations Research*, 52 :422–439.

[Barrett et al., 2000] Barrett, C., Jacob, R., and Marathe, M. (2000). Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3) :809–837.

[Basnal et al., 2015] Basnal, P., Singh, V., Shukla, A., Kumar, D., and Kadam, A. P. A. (2015). Demand responsive transport.

[Bast, 2009] Bast, H. (2009). *Car or public transport-two worlds*. Springer.

[Bast et al., 2010] Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., and Viger, F. (2010). Fast routing in very large public transportation networks using transfer patterns. In *Algorithms–ESA 2010*, pages 290–301. Springer.

[Bast et al., 2009] Bast, H., Funke, S., and Matijevic, D. (2009). Ultrafast shortest-path queries via transit nodes. *The Shortest Path Problem : Ninth DIMACS Implementation Challenge*, 74 :175–192.

[Bauer et al., 2011] Bauer, R., Delling, D., and Wagner, D. (2011). Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1) :38–52.

[Bellman, 1956] Bellman, R. (1956). On a routing problem. Technical report, DTIC Document.

[Berbeglia et al., 2010] Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1) :8–15.

[Bit-Monnot et al., 2013] Bit-Monnot, A., Artigues, C., Huguet, M.-J., and Killijian, M.-O. (2013). Carpooling : the 2 synchronization points shortest paths problem. In *OASIcs-OpenAccess Series in Informatics*, volume 33. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Burris and Winn, 2006] Burris, M. W. and Winn, J. R. (2006). Slugging in houston-casual carpool passenger characteristics. *Journal of Public Transportation*, 9(5) :23.

[Cherkassky et al., 1996] Cherkassky, B. V., Goldberg, A. V., and Radzik, T. (1996). Shortest paths algorithms : Theory and experimental evaluation. *Mathematical programming*, 73(2) :129–174.

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

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

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

[Deleplanque and Quilliot, 2015] Deleplanque, S. and Quilliot, A. (2015). Robustness tools in dynamic dial-a-ride problems. In *Recent Advances in Computational Optimization*, pages 35–51. Springer.

[Delling et al., 2009] Delling, D., Pajor, T., and Wagner, D. (2009). Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*, pages 182–206. Springer.

[Deo and Pang, 1984] Deo, N. and Pang, C.-Y. (1984). Shortest-path algorithms : Taxonomy and annotation. *Networks*, 14(2) :275–323.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271.

[Donovan and Work, 2015] Donovan, B. and Work, D. B. (2015). Using coarse gps data to quantify city-scale transportation system resilience to extreme events. *arXiv preprint arXiv :1507.06011*.

[d'Orey et al., 2012] d'Orey, P. M., Fernandes, R., and Ferreira, M. (2012). Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 140–146. IEEE.

[Drews and Luxen, 2013] Drews, F. and Luxen, D. (2013). Multi-hop ride sharing. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search*, pages 71–79,.

[Efentakis and Pfoser, 2013] Efentakis, A. and Pfoser, D. (2013). Optimizing landmark-based routing and preprocessing. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, page 25. ACM.

[Eisner et al., 2011] Eisner, J., Funke, S., Herbst, A., Spillner, A., and Storandt, S. (2011). Algorithms for matching and predicting trajectories. In Müller-Hannemann, M. and Werneck, R. F. F., editors, *ALENEX*, pages 84–95. SIAM.

[Fredman and Tarjan, 1987] Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3) :596–615.

[Furuhata et al., 2013] Furuhata, M., Dessouky, M., Brunet, F. O. M., Wang, X., and Koenig, S. (2013). Ridesharing : The state-of-the-art and future directions. *Transportation Research Part B : Methodological*, 57 :28–46.

[Gallo and Pallottino, 1988] Gallo, G. and Pallottino, S. (1988). Shortest path algorithms. *Annals of Operations Research*, 13(1) :1–79.

[Gandibleux et al., 2006] Gandibleux, X., Beugnies, F., and Randriamasy, S. (2006). Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function. *4OR*, 4(1) :47–59.

[Geisberger, 2010] Geisberger, R. (2010). Contraction of timetable networks with realistic transfers. In *Experimental Algorithms*, pages 71–82. Springer.

[Geisberger et al., 2010] Geisberger, R., Luxen, D., Neubauer, S., Sanders, P., and Volker, L. (2010). Fast detour computation for ride sharing. In *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems - ATMOS'10*, pages pp. 88–99. Th. Erlebach, M. Lubbecke.

[Geisberger et al., 2012] Geisberger, R., Sanders, P., Schultes, D., and Vetter, C. (2012). Exact routing in large road networks using contraction hierarchies. *Transportation. Science*, 46 :388–404.

[Goldberg and Harrelson, 2005] Goldberg, A. V. and Harrelson, C. (2005). Computing the shortest path : A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics.

[Goldberg and Werneck, 2005] Goldberg, A. V. and Werneck, R. F. F. (2005). Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40.

[Häme, 2011] Häme, L. (2011). An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. *European Journal of Operational Research*, 209(1) :11 – 22.

[Hart et al., 1968] Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Sci. and Cyb*, 4 :100–107.

[Herbawi and Weber, 2011] Herbawi, W. and Weber, M. (2011). Evolutionary multiobjective route planning in dynamic multi-hop ride-sharing. In *voCOP'11*, pages 84–95.

[Herbawi and Weber, 2012a] Herbawi, W. and Weber, M. (2012a). Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms. In *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, volume 1, pages 89–96. IEEE.

[Herbawi and Weber, 2012b] Herbawi, W. and Weber, M. (2012b). The ridematching problem with time windows in dynamic ridesharing : a model and a genetic algorithm. In *Proceedings ACM Genetic and Evolutionary Computation Conference (GECCO)*, pages 1–8.

[Hilger et al., 2009] Hilger, M., Köhler, E., Möhring, R. H., and Schilling, H. (2009). Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem : Ninth DIMACS Implementation Challenge*, 74 :41–72.

[Hoffman et al., 2013] Hoffman, K. L., Padberg, M., and Rinaldi, G. (2013). Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer.

[Huguet et al., 2013] Huguet, M.-J., Kirchler, D., Parent, P., and Wolfler Calvo, R. (2013). Efficient algorithms for the 2-way multi modal shortest path problem. *Electronic Notes in Discrete Mathematics*, 41 :431–437.

[Hyytiå et al., 2010] Hyytiå, E., Håme, L., Penttinen, A., and Sulonen, R. (2010). Simulation of a large scale dynamic pickup and delivery problem. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 77 :1–77 :10, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Kieritz et al., 2010] Kieritz, T., Luxen, D., Sanders, P., and Vetter, C. (2010). Distributed time-dependent contraction hierarchies. In *Experimental Algorithms*, pages 83–93. Springer.

[Kim et al., 2007] Kim, S.-W., Won, J.-I., Kim, J.-D., Shin, M., Lee, J., and Kim, H. (2007). Path prediction of moving objects on road networks through analyzing past trajectories. In Apolloni, B., Howlett, R., and Jain, L., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4692 of *Lecture Notes in Computer Science*, pages 379–389. Springer Berlin Heidelberg.

[Kirchler et al., 2011] Kirchler, D., Liberti, L., Pajor, T., and Wolfler Calvo, R. (2011). Unialt for regular language contrained shortest paths on a multi-modal transportation network. In *ATMOS*, pages 64–75.

[Lauther, 2006] Lauther, U. (2006). An experimental evaluation of point-to-point shortest path calculation on roadnetworks with precalculated edge-flags. *The Shortest Path Problem : Ninth DIMACS Implementation Challenge*, 74 :19–40.

[Lenstra and Kan, 1981] Lenstra, J. K. and Kan, A. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2) :221–227.

[Liu et al., 2010] Liu, C.-L., Jou, E., and Lee, C.-H. (2010). Analysis and prediction of trajectories using bayesian network. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 7, pages 3808–3812.

[Lozano and Storchi, 2001] Lozano, A. and Storchi, G. (2001). Shortest viable path algorithm in multimodal networks. *Transportation Research Part A : Policy and Practice*, 35(3) :225–241.

[Ma and Wolfson, 2013] Ma, S. and Wolfson, O. (2013). Analysis and evaluation of the slugging form of ridesharing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 64–73. ACM.

[Ma et al., 2013] Ma, S., Zheng, Y., and Wolfson, O. (2013). T-share : A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 410–421. IEEE.

[Mote and Whitestone, 2011] Mote, J. E. and Whitestone, Y. (2011). The social context of informal commuting : Slugs, strangers and structuration. *Transportation Research Part A : Policy and Practice*, 45(4) :258–268.

[Orda and Rom, 1990] Orda, A. and Rom, R. (1990). Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3) :607–625.

[Pyrga et al., 2008] Pyrga, E., Schulz, F., Wagner, D., and Zaroliagis, C. (2008). Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 12 :2–4.

[Sanders and Schultes, 2005] Sanders, P. and Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms*, volume 3669 of *LNCS*, pages 568–579. Springer.

[Sanders and Schultes, 2006] Sanders, P. and Schultes, D. (2006). Engineering highway hierarchies. In *Algorithms–ESA 2006*, pages 804–816. Springer.

[Sanders and Schultes, 2007] Sanders, P. and Schultes, D. (2007). Engineering fast route planning algorithms. In *Experimental Algorithms*, pages 23–36. Springer.

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

[Savelsbergh and Sol, 1995] Savelsbergh, M. W. and Sol, M. (1995). The general pickup and delivery problem. *Transportation science*, 29(1) :17–29.

[Schultes, 2005] Schultes, D. (2005). Fast and exact shortest path queries using highway hierarchies. *Master-Arbeit, Universität des Saarlandes, Saarbrücken*.

[Schultes and Sanders, 2007] Schultes, D. and Sanders, P. (2007). Dynamic highway-node routing. In *Experimental Algorithms*, pages 66–79. Springer.

[Schulz, 2005] Schulz, F. (2005). *Timetable information and shortest paths*. PhD thesis, Karlsruhe, Univ., Diss., 2005.

[Schulz et al., 2000] Schulz, F., Wagner, D., and Weihe, K. (2000). Dijkstra's algorithm on-line : an empirical case study from public railroad transport. *Journal of Experimental Algorithmics (JEA)*, 5 :12.

[Stiglic et al., 2015] Stiglic, M., Agatz, N., Savelsbergh, M., and Gradisar, M. (2015). The benefits of meeting points in ride-sharing systems. *ERIM Report Series Reference No. ERS-2015-003-LIS*.

[Tao, 2007] Tao, C.-C. (2007). Dynamic taxi-sharing service using intelligent transportation system technologies. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pages 3209–3212.

[Varone and Aissat, 2015] Varone, S. and Aissat, K. (2015). Multi-modal transportation with public transport and ride-sharing - multi-modal transportation using a path-based method. In *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 1, Barcelona, Spain, 27-30 April, 2015*, pages 479–486.

[Wagner et al., 2005] Wagner, D., Willhalm, T., and Zaroliagis, C. (2005). Geometric containers for efficient shortest-path computation. *Journal of Experimental Algorithmics (JEA)*, 10 :1–3.

[Wolfler Calvo et al., 2004] Wolfler Calvo, R., Luigi, F., Haastrup, P., and Maniezzo, V. (2004). A distributed geographic information system for the daily car pooling problem. *Computers & Operations Research*, 31 :2263–2278.

[Ziliaskopoulos and Wardell, 2000] Ziliaskopoulos, A. and Wardell, W. (2000). An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research*, 125(3) :486–502.

Résumé

Le besoin de se déplacer est un besoin fondamental dans la vie de tous les jours. Avec l'extension continue des zones urbaines, l'augmentation de la population et l'amélioration du niveau de vie des citoyens, le nombre de voitures ne cesse d'augmenter. Ceci étant, la plupart des transports publics proposés aujourd'hui obéissent à des règles qui manquent de souplesse et qui incluent rarement le caractère dynamique, en temps et en espace, de la demande. Cela réduit ainsi l'attractivité de ces services et les rendant même parfois difficilement supportables. De ce fait, la majorité des usagers utilisent encore leur propre véhicule. Ce grand nombre de véhicules, qui est en augmentation continue sur les réseaux routiers, provoque de nombreux phénomènes de congestion induisant une surconsommation de carburant, des émissions inutiles de gaz à effet de serre et une perte de temps importante. Pour y remédier, nous proposons dans cette thèse de nouveaux systèmes de déplacement des usagers avec différents modèles d'optimisation pour la mobilité partagée (covoiturage et taxis-partagés) ainsi que la combinaison de la mobilité partagée avec les transports publics.
Les expérimentations sont réalisées sur de vrais réseaux routiers ainsi que sur des données réelles. Ces nouveaux systèmes améliorent considérablement la qualité de service des systèmes classiques existants en termes de coût et de flexibilité tout en ayant un temps de calcul raisonnable.

Mots-clés: Transport - Mobilité partagée - Optimisation - Graphe - Problème de plus court chemin

Abstract

The travelling is a fundamental part of everyday life. The continuous expansion of urban areas combined with the population increasing and the improvement of life standards increases the need of mobility and the use of private cars. Furthermore, the majority of public transportations are subject to rules lacking of flexibility and rarely taking into account the dynamic context. The attractiveness of public transportation is therefore reduced and, as a consequence, its financial support, resulting in a further deterioration of the public services quality and flexibility. Therefore, the majority of users still use their own vehicles. The number of vehicles is continuously increasing on road networks causing important phenomena of congestion, high fuel consumption and emissions of greenhouse gases, time loss. This unpleasant situation forces communities to consider alternative solutions for the mobility such as ride-sharing, an interesting alternative to solo car use.
The overall objective of this thesis is to propose new travel systems for users through the introduction of optimization models for shared mobility (ride-sharing and taxi-sharing) and the combination of shared mobility and public transportation. The computational experiments are carried out on real road networks and real data. Our numerical results show the effectiveness of our approach, which improves the quality of service compared to the traditional systems in terms of cost and flexibility. The running time remains reasonable to allow using our framework in real-time transportation applications.

Keywords: Transport - Shared mobility - Optimization - Graph - Shortest path problem