Calcul d'itinéraires multiples et de trajets synchronisés dans des réseaux de transport multimodaux

Gregoire Scano

Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue par :
Grégoire Scano

le 8 septembre 2016

Titre :

Calcul d'itinéraires multiples et de trajets synchronisés
dans des réseaux de transport multimodaux

École doctorale et discipline ou spécialité :
EDSYS : Informatique 4200018

Unité de recherche :
LAAS-CNRS

Directeur/trice(s) de Thèse :

Marie-José Huguet, Maître de Conférences HDR, INSA Toulouse
Sandra Ulrich Ngueveu, Maître de Conférences, INP Toulouse

Jury :

Christelle Guéret-Jussien, Professeur des Universités, Université d'Angers, Rapporteur
Ammar Oulamara, Professeur des Universités, Université de Lorraine, Rapporteur
Nour-Eddin El Faouzi, Directeur de Recherche, IFSTTAR, Examinateur
Philippe Lacomme, Maître de Conférences, ISIMA Clermont-Ferrand, Examinateur
Emmanuel Néron, Professeur des Universités, Université de Tours, Examinateur
Julien Lesbegueries, Docteur, Ingénieur MobiGIS, Invité

Abstract

Efficiency and simplicity are two conditions upon which the use of a transportation system is relevant. May it be intentional or imposed, an increasing mobility triggers the need to enhance the transportation offer. In turn, such a response encourages an even more demanding mobility in a constantly adapting cycle. In parallel, new and forthcoming means of transportation emerge from time to time with unknown practices and renewed actors : exactly like what carpooling is stirring at the moment. Passenger information systems can technically deal with such evolutions thanks to improved technologies but they still struggle to keep up with constantly changing usage expectations.

From this perspective the computation of several paths from an origin to a destination becomes increasingly relevant. This issue is even more crucial in dense transportation networks in which many modes and lines of transportation are combined. Indeed, giving some travelling choices to the end user reduces the feeling of exclusion, anxiety and the lack of understanding which may arise when facing arbitrary decisions dictated by a software or an internet application. It is also helpful to estimate the quality of the transportation offer since the more paths exist to go from point A to point B within a fixed time window, the better the service is. This thesis focuses on the computation of such alternatives by the gradually increasing enumeration of paths between two points. Given this input, the pruning necessary to obtain such a diverse selection is assumed not to be known in advance. It is left up to transportation professionals who may choose a fitted solution based on their specific knowledge and objectives.

Another subject studied in this thesis concerns the itinerary synchronisation of several users for various social uses such as shared travels. It is here seen from the perspective of carpooling. Considering only two users, the problem is to minimise the travelling cost of the users under the constraint that they must share some part of their respective trips with one another. Solving this problem is equivalent to finding a pick up point and a drop off location between which both paths overlap. Multiple corner cases concerning the transportation conditions of each user as well as the special cases of shared origins or destinations are studied. The constraints on the arrival and/or departure times may also vary. Last but not least and since the driver is often penalized when giving up a lift, the restriction to a maximal detour the driver accepts, compared to his shortest path, is analysed with respect to the benefits such a limitation generates.

This thesis was funded by the MobiGIS company under the CIFRE (Industrial Agreement of Training through Research) researching context. The related work consisted in the practical implementation of mobility solutions within the framework of the company as well as the experimental performances evaluation of the algorithms proposed to solve them.

Résumé

L'utilisation des réseaux de transport est conditionnée par l'efficacité et la simplicité de leur utilisation. En réponse à une mobilité exacerbée, volontaire ou subie, l'offre de transport se développe et motive tout à la fois, en un cycle continu, des déplacements encore plus exigeants. De manière complémentaire, la mobilité est bousculée par l'arrivée de nouvelles modalités de transport pouvant faire émerger, comme dans le cadre du covoiturage, des acteurs ou des pratiques jusqu'alors inexistants. Si la technologie permet de suivre cette évolution dans les services d'information aux voyageurs, il reste toujours à satisfaire des attentes déterminées par des usages en constante évolution.

C'est de ce point de vue que l'obtention de chemins multiples pour relier une origine à une destination est un facteur qui n'est plus à négliger, surtout dans des réseaux de transport denses et comportant de nombreux modes et lignes de transport. Une liberté dans le choix laissé à l'utilisateur du réseau réduit les sentiments d'exclusion, d'incompréhension ou d'anxiété qui peuvent survenir face à une application logicielle ou sur internet et qui effectuent des choix arbitraires de façon autoritaire. De plus, cela permet de vérifier la qualité de l'offre de transport, car plus il existe de moyens différents pour effectuer un trajet dans un intervalle de temps donné, meilleur est le service. Cette thèse s'intéresse au calcul de telles alternatives par le biais de l'énumération par coût croissant des chemins entre deux points, puis par le filtrage de ceux-ci suivant des critères, supposés quelconques et laissés à l'appréciation des professionnels de transport qui peuvent ainsi faire varier les angles d'analyses de leurs offres.

Par ailleurs, la synchronisation de trajets de plusieurs utilisateurs, en vue d'usages sociaux ou de déplacements mutualisés, est étudiée dans ce manuscrit sous l'angle du covoiturage. En ne considérant que deux usagers, l'objectif est de minimiser le temps de trajet global des participants sous la contrainte qu'ils partagent une partie de leur chemin entre un point de rencontre et un point de séparation qu'il faut alors déterminer. Sont également étudiées les variantes associées au changement des conditions de transport de chacun des participants comme l'établissement d'une origine ou d'une destination commune parallèlement à des contraintes sur les heures de départ ou d'arrivée des usagers. Enfin, puisque la voiture est très souvent pénalisée par la prise en charge d'un piéton, il convient d'étudier comment ce détour peut être contraint et les impacts sur les gains que cette limitation engendre.

Cette thèse a été réalisée dans un contexte CIFRE pour la société MobiGIS. Les travaux qui s'y rapportent ont fait l'objet de réalisations pratiques tant pour fournir des solutions de mobilité dans le cadre des activités de l'entreprise que pour évaluer expérimentalement les performances des algorithmes proposés pour les résoudre.

Remerciements

Je souhaite conclure ces trois années et demi de travail, parfois grisant, souvent difficile, par de sincères remerciements à Marie-José et à Sandra pour leur constante implication, leur professionalisme et leur bienveillance : à toutes deux mes sentiments les plus chaleureux.

Merci à MobiGIS ainsi qu'à l'équipe ROC du LAAS-CNRS de m'avoir permis d'effectuer cette thèse dans les meilleures conditions possibles. Je veux ici témoigner ma reconnaissance du temps et de l'énergie que les personnes y ayant travaillé durant la thèse ont pu y consacrer pour qu'au delà des difficultés que nous avons rencontrées cette thèse soit enfin achevé.

Je remercie Christèle Guéret-Jussien et Ammar Oulamara pour avoir accepté de rapporter ce manuscrit dans une période courte et estivale ainsi que les examinateurs Nour-Eddine El Faouzi et Philippe Lacomme tout comme le président du jury Emmanuel Néron pour l'intérêt qu'ils ont porté au travail qui y est présenté, enfin Julien pour s'être impliqué jusqu'à ce dénouement.

J'ai également des pensées émues envers tous mes collègues pour les moments que nous avons partagés, toujours conviviaux et appréciés.

Table des matières

Chapitre 1

La mobilité, usages et besoins

Si la connaissance d'un cheminement terrestre reste encore souvent heuristique par l'utilisation de ce qui est quotidiennement appelé le sens de l'orientation, le recours à une carte s'avère souvent nécessaire. En effet, elle est un outil de l'intelligence du territoire et une mobilité informée et efficace en est sa principale motivation. Elle permet de transmettre et d'étoffer ce savoir spatial tout en proposant à son utilisateur une réponse de qualité et performante lors de son utilisation.
C'est ainsi désormais que la modélisation informatique permet d'optimiser des trajets, en temps, en distance ou relativement à tout autre mesure de façon plus ou moins aisée en fonction de la calculabilité et la complexité des problèmes de cheminement posés. De façon parallèle, l'information cartographique numérique permet de fusionner la toujours grandissante quantité de moyens de transport pour l'actualiser et la restituer en une information concentrée plus simple d'accès et d'utilisation.
Faut-il encore espérer que cette multimodalité apporte des solutions à des soucis de congestion des infrastructures existantes tout en faisant face à des espoirs de mobilité toujours plus ambitieux, fussent-ils irraisonnés.
Car si cette montée en puissance de l'information voyageur est rendue possible par la technologie elle répond également à l'augmentation des besoins de mobilité. Elle se confronte désormais à de nouveaux usages dont les pratiques sortent des systèmes centralisés et qui posent donc de nouveaux défis en vu de leur intégration avec des modes plus classiques. En particulier parce qu'ils ne sont plus uniquement planifiables à l'avance mais s'établissent en des usages spontanés et imprévisibles.

C'est dans cette optique de promotion de déplacements efficaces mais néamoins responsables que s'inscrit cette thèse. Dans la continuité des travaux précédents menés au LAAS-CNRS [18, 26] et portés par la société MobiGIS [41], elle vise à intégrer la diversité des modes de transport actuels à des besoins de mobilité nouveau comme le covoiturage. Au delà de l'information voyageur dans des applicatifs clients, le développement d'outils d'aide à la décision dans le domaine du transport public urbain est une préoccupation tout aussi importante que le déplacement car elle le conditionne en amont. Prises par des professionnels du transport et de la cartographie, ces décisions sont partiellement basées sur des calculateurs qui ne résolvent pas le problème de façon globale mais uniquement sur des problématiques locales de déplacement. Ainsi cela offre toute latitude aux décisions métiers, politiques et de bon sens qu'il n'est pas nécessairement possible ou utile

d'intéger partiellement ou en totalité dans des modèles mathématiques et numériques. Cependant, au moment de l'utilisation d'une offre de transport l'utilisateur n'est pas toujours confiant face à la machine dans les choix qui par elle lui sont imposés. Une partie du travail présenté dans ce manuscrit consiste donc à trouver des alternatives au plus court chemin dans le but de réduire cet aspect frontal à la machine par la multiplicité des choix qu'elle permet alors. Par le biais du calcul de plusieurs chemins ordonnés par leur coût, une sélection des déviations possibles pour éviter à certains usagers l'utilisation ou l'enchaînement de plusieurs modes est opérée. Dans cette optique, il est supposé dans ces travaux que la qualification de ce qui constitue une différence entre deux chemins multimodaux n'est pas à laisser au concepteur de l'algorithme mais à ses utilisateurs directs, les usagers, et indirects, les autorités organisatrices de transports, qui les mettent à disposition des premiers.
Un autre aspect d'une mobilité raisonnée et assumée correspond au covoiturage car il noue une relation étroite entre deux modes auparavant mis dos à dos, à savoir la voiture personnelle et les transports en commun. Des relations entre offres et demandes de transport doivent s'établir sur des critères objectifs et numériques de temps et de gains de parcours mais également concient de facteurs humains comme l'ambiance et le confort du trajet. Les travaux s'y rapportant, présentés dans cette thèse, supposent toujours que le covoiturage est une décision active du conducteur pour consacrer son véhicule ainsi qu'une partie de son temps à un déplacement gracieux et plus collectif qu'une utilisation individuelle ou familiale.

Dans les contributions scientifiques de ce travail s'inscrit la présentation d'un article sur les k plus courts chemins à la conférence IESM 2015. Quant à l'exploration des variantes liées au problème de synchronisation des trajets de deux covoitureurs et ses avancées feront, je l'espère, l'objet d'un article prochain.

Pour aborder ces problèmatiques et contributions, le chapitre 2 de ce manuscrit commence par définir les notions nécessaires à l'expression de problèmes de plus courts chemins. Pour cela la structure de données de graphe est rappelée, structure sur laquelle des cheminements peuvent être calculés. Puis les problématiques de calcul de plus court chemin sont énumérées conjointement avec les algorithmes de la littérature permettant de les résoudre efficacement. S'en suivent la description de problèmes de synchronisation de trajets multiples comme c'est le cas pour des trajets aller-retour, par exemple pour faire des trajets domicile-travail sur lesquels un véhicule personnel ne peut être utilisé que sur une partie du trajet. Enfin, différentes approches de résolution de trajets alternatifs sont présentées.

Abordé dans le chapitre 3, le calcul de k plus courts chemins, fussent-ils avec ou sans cycles, sous contraintes et sur des graphes de transport dépendants du temps, est abordé par des adaptations, parfois infructeuses, de certains algorithmes de la litérature

alors détaillés. Y est également introduit une nouvelle méthode permettant le calcul de ces chemins sans cycles par l'énumération puis le filtrage de ceux en contenant, plutôt que de chercher des chemins élémentaires immédiatement. Cette approche n'a jamais, à notre connaissance, fait l'objet de mesures expérimentales visant à mettre en balance la concurrence des deux approches avec ou sans cycles dans ce contexte des réseaux de transport multimodaux.

Le chapitre 4 est dédié à l'étude des situations de recherche de points de synchronisation optimaux pour le covoiturage. Dans le cas où un passager désire profiter d'une offre de covoiturage d'un conducteur mettant à disposition son temps et son véhicule, il convient de déterminer, si cela est possible, le point de rencontre et le point de séparation de ces deux participants minimisant le temps de trajet total. En particulier, la restriction du détour effectué par le conducteur par rapport à son trajet optimal est étudiée au regard du gain qu'entrainerait pour le piéton un tel détour.

Enfin, ce manuscrit se termine dans le chapitre 5 par une description succinte des activités de transfert et de développement industriel effectués en parallèle de ces travaux de recherche. Se déroulant en majeure partie à MobiGIS, où en confrontation directe avec une utilisation client, les projets `MobiAnalyst`, `MobiAnalyst SaaS/Cloud` et `Moveazy` ont concentré la majeure partie du travail, ces activités ont également été présentes au LAAS-CNRS, dans le cadre du développement de la plateforme *PlayMob'*, par l'intégration au calculateur des algorithmes de calcul de plus courts chemins implémentés pendant la thèse.

Ce travail se conclut sur plusieurs résultats et ouvre quelques perspectives nouvelles dans le cadre de la recherche de chemins alternatifs ou synchronisés.

Chapitre 2

Cheminements dans les graphes

Le traitement automatisé de tâches par les ordinateurs est effectué par la manipulation d'objets et de concepts issus d'une modélisation des problèmes qu'ils doivent résoudre. Cette transposition de la réalité s'établit à partir des briques de base que constituent les structures de données pour former des objets plus complexes avec lesquels des algorithmes appropriés pourront résoudre le problème posé. Parmis ces dernières, les graphes sont des structures permettant un niveau d'abstraction élevé qui peut être enrichi en fonction des besoins. Ils se trouvent donc au cœur de problèmes et notamment des cheminements, de leurs alternatives et synchronisations.

2.1 Structure de données

2.1.1 Graphes

Un **graphe** est un objet mathématique discret établissant des relations entre un ensemble d'éléments. Il permet d'abstraire des situations dans lesquelles des liens entre composants peuvent être définis. Dans notre cas, la relation nouée entre un ensemble de positions géographiques et la façon de se déplacer de l'une à l'autre modélise les réseaux de transport, qui peuvent donc être transcrits en graphes (voir Appendice A) afin d'effectuer des calculs de cheminement dessus.
Formellement, un **graphe** G est un triplet (V, E, γ) formé d'un ensemble V de **nœuds** ou **sommets**, d'un ensemble E d'**arcs** ou **arêtes**, et d'une relation $\gamma : E \to V \times V$ entre V et E. Il peut y avoir plusieurs arêtes entre deux nœuds tandis que certains nœuds peuvent être **déconnectés** puisque reliés par aucune arête à d'autres nœuds. Dans le reste du manuscrit les ensembles V et E seront toujours finis et dans ce cas le graphe est dit **fini**.
De plus, si pour tout couple de sommets, l'existence d'un arc d'un nœud vers un autre implique l'existance d'un arc du deuxième nœud vers le premier alors le graphe est dit **non-orienté** et est alors composé d'arêtes et de sommets, autrement il est dit **orienté** et les ensembles V et E contiennent les nœuds et les arcs respectivement. C'est le cas des graphes de transport pour lesquels un seul sens de circulation peut exister entre certaines paires de sommets.
Il est possible de construire une famille de fonctions donnant pour chaque sommet ceux qui lui sont **adjacents**, c'est à dire tels qu'il existe un arc de E pour lequel le couple obtenu par l'application de γ le contient. Si le sommet est le premier élément du couple

il sera appelé **prédécesseur** et s'il s'agit du second élément il sera appelé **successeur**. Pour chaque sommet l'accès à ses ensembles de ses prédécesseurs et de ses successeurs est à la base de toute exploration du graphe à partir de ce sommet. La cardinalité de ces ensembles définit les différents **degrés** d'un sommet.
Un graphe est dit **fortement connexe** s'il existe un chemin entre tous ses couples de nœuds. Les graphes de transport considérés seront toujours supposés fortement connexes. En effet les problématiques de mobilité sont établies sur des éléments qui sont reliés les uns aux autres. Cependant dans la réalité, cette connexité n'est pas toujours atteinte puisque la génération de graphes de transport est sujette à des erreurs de saisies. Pour palier ce problème il est possible de retirer les nœuds qui ne sont pas connectés ou de les connecter en utilisant des projections spatiales aux nœuds ou aux arcs les plus proches.
Dès lors que γ est bijective et qu'il n'existe pas d'arcs d'un sommet vers lui même, le graphe est dit **simple**. La notation simplifiée $G = (V, E)$ est donc adoptée et il existe au plus un arc entre tout couple de sommets différents et aucun arc entre tout couple de sommets identiques. Nous considérerons, pour simplifier, que les graphes de transport sont des graphes simples bien qu'il puisse exister deux modes de transport différents entre deux sommets. Lorsque ce cas se présente, il est possible de créer autant de nœuds fictifs qu'il y a d'arcs entre deux sommets, de relier le sommet origine à chacun d'eux puis de les relier tous au nœud cible.
Enfin, la **densité** d'un graphe est donnée par le rapport du nombre d'arcs sur le nombre maximal que peut en comporter le graphe. Puisque les graphes considérés sont désormais simples et orientés le nombre maximal d'arcs qu'ils peuvent comporter est le nombre de nœuds au carré. La densité dans les réseaux de transport est en général proche de 4 car un nœud ne peut connecter que les nœuds qui lui sont voisins géographiquement.

2.1.2 Graphes étiquetés

Les graphes **étiquetés** permettent de rajouter de l'information sur chaque arc ou sommet en plus des relations que décrivent les arcs entre les sommets.
Cet ajout d'information se fait par l'adjonction d'autant de fonctions que nécessaire et qui chacune donne à tout arc ou nœud une valeur dans un ensemble arbitraire. Lorsque ces ensembles sont composés de nombres la notion de graphe **valué** sera préférée. Par exemple sur un graphe de transport, une fonction donnera la longueur à parcourir, voire le temps qu'il faut à pieds ou en voiture, pour relier un nœud à un autre. Mais il est également possible de spécifier le mode de transport de l'arc, qu'il s'agisse d'un bus, d'un métro ou d'un arc de voirie, ce qui est une donnée d'importance dans ce manuscrit.

2.1.3 Chemins

Un **chemin** sur un graphe est une séquence d'arcs consécutifs c'est à dire tels que chaque nœud successeur soit le nœud prédécesseur de l'arc suivant, sauf pour le prédécesseur du premier arc et le successeur du dernier. Si le chemin ne passe pas deux

fois par le même arc il est dit **simple** alors que s'il ne contient pas deux fois le même nœud, il est dit **élémentaire** ; tout chemin élémentaire est donc simple. Les ensembles des chemins simples et élémentaires d'un graphe sont de taille finie mais de cardinalité factorielle dans le pire des cas en fonction du nombre de nœuds et d'arcs.
Lorsque les graphes sont valués, il est possible de construire une fonction de **valuation** d'un chemin qui prend en entrée un ensemble de fonctions de valuations du graphe, un arc et un ensemble de valeurs chacune dans un ensemble arbitraire, et qui retourne un nouvel ensemble de ces valeurs mises à jour après le passage par l'arc donné. Par exemple si le graphe dispose d'une fonction donnant le coût de chaque arc alors la fonction d'évaluation des chemins prendrait en entrée un arc, la fonction permettant de récupérer le coût d'un arc et le coût courant du chemin. La fonction d'évaluation des chemins retournera ensuite le nouveau coût mis à jour après le parcours de l'arc. Il est donc possible d'obtenir un **chemin valué** sur un graphe valué.

2.1.4 Cheminements

Les problèmes de **cheminement** consistent à trouver des chemins entre deux nœuds sur un graphe étiqueté ayant certaines propriétés vis à vis de fonctions de valuation de chemins. La version la plus simple est de trouver un chemin qui minimise le nombre de nœuds qu'il traverse pour aller de l'origine à la destination. En travaillant sur des graphes valués, le problème de plus court chemin classique cherche plutôt à minimiser le coût du trajet. Il est ensuite possible de complexifier à l'envie par l'ajout de contraintes et d'objectifs. Des algorithmes spécialisés peuvent être élaborés pour chaque type de problème.

2.2 Problèmes du plus court chemin

Le calcul d'un plus court chemin est usuellement posé sur un graphe valué par une unique fonction de coût. Dans le cadre de ce manuscrit les fonctions de valuations seront à valeurs dans les réels positifs et le coût d'un chemin consiste à effectuer la somme du coût des arcs qui le composent. Il s'agit d'un problème bien identifié et très classique en informatique.
La structure de données utilisée y est vue comme une abstraction du problème concret et l'information disponible pour résoudre le problème du cheminement le plus court se limite à la description du graphe et de ses fonctions de valuation.
Usuellement lorsque le problème de plus court chemin est évoqué il s'agit de trouver le plus court chemin entre deux nœuds. Toutefois, cela peut être étendu au calcul depuis un nœud vers chacun des autres ce qui se résume par le calcul d'un arbre de plus courts chemins. Enfin, il est parfois nécessaire d'obtenir un plus court chemin entre tout couple de nœuds ce qui s'apparente à du calcul intensif pour lequel il existe des algorithmes dédiés qui ne seront pas abordés dans cet ouvrage.

2.2.1 Algorithmes classiques

Les algorithmes permettant de résoudre efficacement le problème de plus court chemin se basent généralement sur la programmation dynamique [24]. Cette technique, qui n'est valide que pour certains types de problèmes et en l'occurence pour un bon nombre de problèmes de cheminements, s'appuie sur le *principe d'optimalité de Bellman* stipulant qu'une solution optimale peut être entièrement déterminée à partir de la solution optimale de chacun de ses sous-problèmes. En commençant par résoudre les problèmes les plus petits, il est possible de déterminer de façon incrémentale les solutions optimales de sous-problèmes de plus en plus grands jusqu'à arriver au problème global à résoudre.

Algorithme de Bellman-Ford-Moore La méthode de Bellman[37][52][25] calcule un arbre de plus courts chemins à partir d'une origine. A chaque itération k, l'arbre obtenu est celui des plus courts chemins contenant au maximum k arcs. Au début le coût de tous les nœuds est fixé à $+\infty$ sauf pour l'origine qui a un coût entier fixé, en général 0. Le plus court chemin de l'origine à elle-même a un coût nul et aucun des autres nœuds n'est accessible, l'arbre se limite donc au nœud origine. A l'itération k tous les coûts des nœuds sont mis à jour en comparant leur valeur courante avec celle obtenue en prolongeant vers eux un des chemins de l'arbre. Ne sont candidats à la prolongation vers la nœud j que les chemins se terminant par un nœud i prédécesseur de j dans le graphe.
La correction de l'algorithme repose sur le fait que les coûts approximatifs, fixés à l'infini au départ, vont être diminués à partir d'un autre nœud et d'un arc qui les relie à chaque fois que cela est possible. Cette relaxation est appliquée à tous les arcs du graphes à chaque itération elle même répétée autant de fois qu'il y a de nœuds dans le graphe moins une fois. La complexité de l'algorithme est donc de $O(|E| \times |V|)$.
En supposant que le graphe puisse contenir des valuations négatives, la méthode de Bellman permet d'identifier des cycles de longueurs négatives atteignables depuis l'origine en identifiant les nœuds dont le coût continue de baisser après V itérations. Si la valuation de certains arcs est négative mais qu'il n'existe pas de cycles de longueurs négatives atteignables depuis l'origine alors l'algorithme est à même de trouver le plus court chemin contrairement à l'algorithme de Dijkstra introduit ci après ne pouvant travailler que sur des graphes à valuations positives. En revanche, s'il n'existe pas de coût négatif alors les deux algorithmes sont applicables et Dijkstra possède une complexité et une efficacité bien meilleure.

Algorithme de Dijkstra L'algorithme de Dijkstra [34] date de 1959 et calcule un arbre de plus courts chemins, c'est à dire que pour un certain nœud d'un graphe, appelé origine, il trouve sous forme d'arbre, le plus court chemin de ce nœud vers tous les autres. Le calcul d'un plus court chemin entre deux nœuds d'un même graphe étant un sous problème de celui résolu par l'algorithme de Dijkstra, il est également possible de

l'utiliser dans ce cas là en arrêtant l'algorithme plus tôt.
L'algorithme de Dijkstra est un algorithme dit à fixation d'étiquettes. En effet, après avoir inséré l'origine dans une liste de nœuds candidats, il sélectionne à chaque itération le nœud de plus petit coût dans cette liste puis l'étend vers chacun de ses successeurs. Si un successeur n'a jamais été dans la liste des nœuds candidats, il y est intégré, alors que s'il y est déjà présent mais avec un coût supérieur au nouveau coût, il est mis à jour (ou dans le pire des cas supprimé puis réajouté). Si au contraire le nœud successeur est déjà présent précédemment dans la liste des nœuds candidats alors il est ignoré, car aucun nœud n'a besoin d'être considéré plusieurs fois.
Dans le problème originel de calcul du plus court chemin d'un nœud vers tous les autres, il faut itérer jusqu'à ce que la liste des nœuds candidats soit vide, tous les nœuds du graphes sont alors dans l'arbre de plus court chemin (si le graphe est connexe ou sinon l'algorithme s'arrête lorsque la liste des nœuds candidats est vide). En revanche, dans le cas d'un plus court chemin point à point, il faut stopper l'algorithme dès que la destination est sélectionnée dans la liste de candidats.
La complexité de l'algorithme est notée $O(|E| \times \mathcal{C}_{stocke} + |V| \times \mathcal{C}_{extrait})$ avec $\mathcal{C}_{stocke}$ la complexité nécessaire au stockage ou à la modification d'un candidat dans la liste et $\mathcal{C}_{extrait}$ la complexité de l'extraction du meilleur candidat dans cette même liste. En effet chacun des nœuds ne peut sortir au plus qu'une seule fois de la liste de candidats et il peut y être modifié autant de fois qu'il y a d'arcs qui lui sont adjacents. En pratique, la liste de candidats sera un tas et la complexité changera en fonction du tas sélectionné (binomial, binaire, fibonacci, etc). La meilleure complexité théorique $O(|E| + |V| \times log(|V|))$ est atteinte lorsque l'implémentation de la liste de candidats est basée sur un tas de fibonacci. Cependant il a souvent été observé dans la pratique qu'un tas binomial bien que de complexité supérieure est souvent plus performant en pratique.
L'algorithme de Dijkstra peut également être utilisé pour calculer un ensemble de nœuds atteignables dans la limite d'un coût fixé depuis une origine ou vers une destination. Dans ce cas, il suffit de modifier la condition d'arrêt de l'algorithme pour stopper les itérations dès qu'un nœud de coût supérieur à la limité fixée est sélectionné. L'ensemble des nœuds obtenus forment ce qu'on appelle un *isochrone* lorsque le coût est un temps.

Algorithme bidirectionnel L'algorithme de Dijkstra explore les nœuds qui sont dans un voisinage de plus en plus éloigné du nœud origine en terme de coût.
L'algorithme bidirectionnel présenté par Nicholson en 1966 [54] est dédié au calcul de plus court chemin point à point entre un nœud origine et un nœud destination. Il effectue simultanément une recherche depuis l'origine en parcourant le graphe dans le sens des arcs, et depuis la destination en parcourant le graphe dans le sens inverse des arcs. L'exploration dans le sens direct depuis l'origine est généralement appelée exploration *avant* ou *forward* et l'exploration dans le sens inverse depuis la destination est généralement appelée exploration *arrière* ou *backward*. Cela peut se faire en traitant alternativement

les candidats d'une liste pour l'exploration avant et les candidats d'une liste pour celle arrière, ou bien en les considérant de façon conjointe dans une même liste. Il y a donc sur chaque nœud un coût depuis l'origine et un autre vers la destination. Par simplicité, nous considérerons l'utilisation de deux listes de candidats. Les coûts sont mis à l'infini au début de l'algorithme sauf pour l'origine et la destination. A chaque itération, un nœud d'une des deux listes de candidats est sélectionné et étendu vers ses nœuds successeurs. La terminaison de l'algorithme se produit lorsque la somme des deux coûts (non infinis) sur un nœud est plus petite que la somme des coûts des meilleurs candidats courants de l'exploration avant et de l'exploration arrière. De cette façon, il est garanti qu'il n'existe pas de nœuds candidats issus des explorations avant ou arrière qui pourraient conduire, par leur rencontre, entre eux ou avec d'autre nœuds, à une valeur de coût plus petite.
La complexité de l'algorithme bidirectionnel est théoriquement identique à celle de l'algorithme de Dijkstra, cependant l'espace de recherche est réduit de façon plus ou moins efficace selon les graphes. En effet, alors que l'algorithme de Dijkstra explore un voisinage dont le rayon en terme de coût est légèrement supérieur au coût du plus court chemin, l'algorithme bidirectionnel visite deux voisinages dont chacun des rayons est supérieur à la moitié du coût du plus court chemin. Par exemple, dans le cas d'un graphe uniforme pour lequel les coûts sont distribués de façon homogène alors l'espace de recherche est environ divisé par 4.

2.2.2 Algorithmes informés

Bien que les algorithmes présentés précédemment soient performants, le temps de résolution devient d'importance lorsqu'ils sont appliqués sur des graphes réels de transport. Pour améliorer les performances expérimentales il est nécessaire, à défaut d'obtenir des algorithmes avec de meilleures complexités temporelles, de disposer d'informations supplémentaires sur le problème à résoudre : on parle d'algorithmes informés. Ils peuvent être divisés en trois catégories suivant qu'ils exploitent :

- une connaissance sur le domaine du problème ;
- un calcul préalable sur le graphe lui même, on parle alors de pré-calculs ;
- ou bien une combinaison des deux.

Algorithme A^* : utilisation du domaine L'algorithme A^* exploite les informations venant du domaine du problème afin de limiter l'espace de recherche. Plus l'information est précise et plus le temps d'exécution sera réduit car moins le nombre d'itérations de l'algorithme sera important. Il permet de calculer un plus court chemin point à point d'une origine vers une destination.
Pour l'algorithme A^*, proposé en 1968 par Hart [42], chaque nœud dispose alors d'une valeur, parfois appelée *heuristique*, estimant le coût jusqu'à la destination. Lors de l'exécution de l'algorithme, la sélection d'un nœud dans la liste de candidats se base alors à la fois sur son coût depuis l'origine (comme pour l'algorithme de Dijkstra) auquel s'ajoute

le coût estimé par l'heuristique. De cette façon, l'exploration du graphe va dans le sens des coût finaux croissants et pas dans celui des coûts courants croissants, ce qui élimine des candidats dont l'évaluation grossière du coût total nous informe qu'il est très peu probable qu'ils engendrent l'optimal.
Si l'évaluation heuristique ne surestime jamais la valeur réelle du plus court chemin entre deux nœuds alors l'algorithme fournit la solution optimale. En revanche, si une valeur était surestimée à un moment donné, elle pourrait être moins prioritaire qu'une autre et cela masquerait alors des candidats menant à l'optimum.
De plus, si l'évaluation est cohérente, c'est à dire qu'elle vérifie l'inégalité triangulaire pour tout triplet de nœuds du graphe, alors de fait chaque nœud n'est traité qu'une seule fois, tout comme dans l'algorithme de Dijkstra, garantissant une complexité polynomiale $O(|E| \times (\mathcal{C}_{stocke} + \mathcal{C}_{heuristique}) + |V| \times \mathcal{C}_{extrait})$ où $\mathcal{C}_{stocke}$ et $\mathcal{C}_{extrait}$ sont respectivement les complexités pour insérer ou modifier un candidat dans la liste de candidats et $\mathcal{C}_{heuristique}$ est le temps nécessaire au calcul de l'heuristique entre deux nœuds. D'ailleurs, dans le cas où l'heuristique est nulle partout, l'algorithme explore autant de nœuds que l'algorithme de Dijkstra et on retrouve la même complexité que cet algorithme.
Enfin, il est montré indépendamment de la cohérence, conformément à l'errata de l'article, qu'il n'existe pas d'autre algorithme admissible à fixation d'étiquettes et à quantité d'information inférieure ou égale qui explore moins de nœuds que l'algorithme A^*. Il n'existe donc pas d'algorithme à fixation d'étiquettes à quantité d'information nulle (donc admissible) meilleur que celui de Dijkstra en tant que cas particulier de l'algorithme A^*.

Algorithme A^* bidirectionnel Ikeda [44] a montré qu'il est également possible d'adapter le coût des arcs d'un graphe appelés alors *coût réduit* en utilisant la fonction heuristique pour faire fonctionner un algorithme de Dijkstra sur ce graphe modifié de la même façon que l'algorithme A^*, c'est à dire en explorant les nœuds dans le même ordre. Toutefois, cela ne peut fonctionner que si l'heuristique est cohérente puisque si tel n'est pas le cas les coûts réduits peuvent alors être négatifs et l'algorithme de Dijkstra ne fonctionne pas sur des graphes dont les valuations sont négatives.
Par la suite, cette formulation permet d'écrire un algorithme A^* bidirectionnel utilisant deux heuristiques indépendantes, une pour l'exploration avant et l'autre pour l'exploration arrière. Une nouvelle heuristique est basée sur l'addition de la moitié de la valeur de chacune des heuristiques avant et arrière. Cette nouvelle valeur heuristique est toujours inférieure à la valeur optimale, l'heuristique est donc admissible ; et additionnée au coût de l'arc d'origine en chaque nœud cela donne une valeur qui est toujours positive puisque chacune des heuristiques est cohérente.
Les propriétés de cette unique heuristique permettent d'appliquer l'algorithme de Dijkstra bidirectionnel sur le graphe dont les coûts sont modifiés à la volée lors des explorations avant et arrière. L'algorithme possède alors le même comportement que l'algorithme A^*.

Toutefois cette nouvelle heuristique est de moins bonne qualité car la division par deux réduit la borne supérieure. L'algorithme explorera donc plus deux nœuds ce qui est compensé par la recherche bidirectionnelle.

Algorithme ALT : utilisation de pré-calculs La propriété démontrée par Ikeda pour utiliser un algorithme A^* bidirectionnel est utilisée par l'algorithme ALT [39] afin d'accélérer la recherche sans disposer d'information provenant du domaine du problème mais uniquement à partir du graphe. Cet algorithme reprend les principes de l'algorithme A^* mais devient bien plus général puisqu'il peut s'appliquer dans n'importe quel contexte pour le calcul d'un plus court chemin entre deux points.
Dans un premier temps un pré-calcul mesure le plus court chemin d'un ensemble de nœuds appelés «repères» vers tous les nœuds du graphe et inversement. Pour cela, il faut déterminer un arbre de plus court chemin en parcours avant mais aussi un autre en parcours arrière. Ensuite, lors de l'exécution de l'algorithme, le coût d'un nœud candidat à la destination est évalué en utilisant ces repères. En effet, en se basant sur l'inégalité triangulaire de façon symétrique, la distance entre un repère et la destination (symétriquement le candidate et le repère) est inférieure ou égale à la somme des distances entre le repère et le candidat (symétriquement la destination et le repère) puis entre ce candidat et la destination. De cette façon, deux bornes supérieures peuvent être obtenues pour le coût du chemin du candidat à la destination pour chacun des repères et il suffit alors de selectionner la plus grande de ces dernières comme valeur de l'heuristique dans l'algorithme. Ainsi plus le nombre de repères est important et plus la probabilité qu'un repère se trouve sur le plus court chemin entre le candidat et la destination est important, auquel cas la valeur est optimale pour ce candidat. A défaut, il faudra faire en sorte que la sélection des repères augmente la valeur des bornes inférieures pour l'ensemble des nœuds du graphe.
Le domaine du problème peut toujours être utilisé comme une borne possible lors du choix du maximum des bornes inférieures mais il peut également être utile pour sélectionner les «repères». En effet plusieurs méthodes existent pour cela incluant les sélections suivantes :

- aléatoire ;
- le plus éloigné, en prenant incrémentalement le nœud le plus loin des repères actuels en terme de distance ou de nombre de sauts ;
- planaire, en découpant le graphe en parts égales autour de son centre et en ajoutant pour chaque portion le nœud le plus éloigné du centre ;
- par évitement, qui cherche à déterminer quelles sont les régions du graphes les moins bien loties par les repères et à ajouter un repère dans ces zones ;
- par couverture, qui utilise la méthode précédente pour effectuer une recherhe locale maximisant le nombre de nœuds pour lesquels les repères sont optimaux, c'est à dire qui donnent la meilleure heuristique.

Autres méthodes Plusieurs méthodes utilisant des pré-calculs existent et sont divisées en deux grandes familles. La première cherche à améliorer le temps de calcul effectif tout en restant modérée sur le temps et l'espace du pré-calcul. Pour les algorithmes de la seconde il est supposé qu'il est possible de déterminer l'ensemble des plus courts chemins d'un graphe en un temps raisonnable pour les stocker temporairement afin d'extraire de cette base de connaissances des indicateurs limités qui pourront être utilisés lors des calculs effectifs.
La méthode la plus connue de la première famille est la contraction hierarchique ([57], [58], [59]). Elle extrait du graphe des propriétés locales à des groupes de nœuds afin d'en dégager des axes majeurs qui seront souvent empruntés pour faire de grandes distances, reléguant les axes secondaires à des recherches locales autour de l'origine et de la destination. De façon pratique, le pré-calcul permet de trouver par un calcul de proximité, quels sont les axes principaux, secondaires et finalement les portions du graphe qui ne seront utilisées qu'en début et fin de parcours pour rejoindre un lieu précis. La construction et le parcours sur un tel graphe, découpé en niveaux, permet de ne pas explorer les axes locaux et focalise la recherche sur les voies permettant une vitesse de déplacement élevée jusqu'à redescendre localement le long de ces axes pour trouver la destination sur les réseaux secondaires.
Dans la deuxième famille se trouvent les algorithmes ([61], [62], [48], [63]) utilisant des drapeaux qui simulent le rôle que jouent les panneaux directifs disposés le long des routes et indiquant un chemin vers des zones qui peuvent être totalement ignorées en fonction de l'endroit courant de la recherche. La construction de conteneurs géographiques partitionnant le graphe en secteurs est à la base de la méthode, les drapeaux sont ensuite le relicat des plus courts chemins calculés depuis chaque nœud du graphe vers tous les autres.
Enfin, l'algorithme SHARC [23] combine les deux approches pour obtenir un algorithme unidirectionnel très efficace.

2.2.3 Prise en compte de la dépendance au temps

Dans les réseaux de transport, la durée de transit le long d'un arc peut dépendre de l'heure à laquelle cet arc est emprunté, notamment pour les arcs représentant le réseau de transport en commun. La modélisation d'un graphe à partir d'un réseau de transport (voir annexe), au niveau des arcs peut se faire de deux façons.
La première, appelée extension temporelle («time-expanded»), consiste à démultiplier le graphe en fonction du temps et de façon plus simple, à multiplier les nœuds à chaque évènement temporel qui s'y déroule. Ainsi pour chaque heure de départ d'un bus, il y aura un nœud dans le graphe le reliant à ses arrêts précédents et suivants en fonction des horaires de départ et d'arrivée sur chacun d'eux respectivement.
La seconde, appelée «dépendente au temps» ou intention/fonction temporelle («time-dependent»), par opposition avec la précédente, ne conserve qu'un seul nœud pour plu-

sieurs horaires mais attribue plusieurs valeurs à l'arc en fonction de l'heure à laquelle il est emprunté. Comparé au modèle précédent, l'information est délocalisée dans des tables horaires et le coût d'un arc devient une fonction du temps, continue par morceaux.
Dans les articles [56] et [32], les deux modèles sont comparés avec un léger avantage pour le modèle dépendent du temps, c'est donc celui qui a été choisi pour le reste de ce manuscrit.

Dans le contexte de graphes dépendant du temps, l'horaire du trajet a une importance lors du calcul d'un plus court chemin puisqu'il détermine le coût des arcs empruntés au fur et à mesure de l'exploration. Lorsqu'un horaire est donné à l'origine, le problème consiste à trouver un plus court chemin arrivant au plus tôt à la destination. Par contre, si un horaire est donné à la destination, il faut alors déterminer un plus court chemin partant au plus tard de l'origine, c'est à dire une heure de départ telle que l'on puisse, en utilisant le chemin solution, arriver à la destination avant l'heure spécifiée. Afin de dénommer ces différents problèmes, Dean [30] a proposé la notation suivante pour caractériser les calculs de plus courts chemins dépendent du temps entre une origine o et une destination d :

— $EA_{o,d}(t)$: calcul de l'arrivée au plus tôt en d en partant de o à l'instant t ;
— $LD_{o,d}(t)$: calcul du départ au plus tard de o pour arriver au plus tard à l'instant t en d.

Si pour un arc, dépendant du temps, il n'est pas possible d'arriver au nœud suivant avec un coût inférieur en partant plus tard, alors l'arc respecte la propriété FIFO («First-In First-Out» ou Premier-entré, Premier sorti) parfois appelée propriété de non-dépassement. Si tous les arcs d'un graphe vérifient cette propriété particulière alors le graphe est dit FIFO puisque cette condition s'applique alors entre tout couple de sommets du graphe.
Sous cette condition, l'algorithme de Dijkstra permet de trouver un plus court chemin entre deux nœuds pour les deux problèmes envisagés et l'ensemble de variantes a été fermé [31]. Dans le cas inverse, il faudrait un algorithme multilabels pour conserver l'ensemble des possibilités sur chaque nœud au cas où un horaire précédant puisse rattraper un horaire suivant.
Bien que l'algorithme de Dijkstra puisse être utilisé, il faut tenir compte du calcul du coût d'un arc suivant l'heure. Puisque les heures sont croissantes et que le réseau est FIFO, alors, les horaires peuvent être triés et l'accès aux données peut être fait en temps linéaire dans le pire des cas, bien que le nombre d'horaires diminuent avec le parcours. Enfin, le nombre d'horaires sur un arc est supposé très inférieur au nombre de nœuds du graphe, la complexité dans le pire des cas reste donc inchangée.

Dépendance au temps et algorithmes bidirectionnels La dépendance au temps va impacter la correction des algorithmes bidirectionnels. En effet, puisque dans les problèmes $EA_{o,d}(t)$ et $LD_{o,d}(t)$ les dates d'arrivées (respectivement de départ), ne sont pas connues, il n'est pas possible de lancer une exploration depuis la destination (respectivement l'origine) et d'obtenir des coûts exacts. Une solution consiste alors à effectuer un calcul en utilisant des bornes supérieures comme proposé par [53] puis d'effectuer une réévaluation faisant appel à ces bornes pour limiter l'espace de recherche.

2.2.4 Intégration de contraintes

Lorsque le plus court chemin que l'on souhaite obtenir doit respecter un certain nombre de contraintes, on parle alors de calcul de plus courts chemins sous contraintes. On peut distinguer deux familles de contraintes, celles correspondant à des ressources qui peuvent varier le long du chemin (énergie du véhicule, temps de parcours piéton) qui ne seront pas étudiées dans ce manuscrit ; et celles correspondant à des contraintes sur les caractéristiques des trajets. En particulier, sur des réseaux de transport, l'intérêt peut porter sur l'utilisation des différents modes de transport. Ainsi, si un passager ne souhaite pas emprunter de bus, cela correspond à une contrainte pour laquelle aucun arc du chemin obtenu ne doit être un arc de bus.
Supposons maintenant que l'utilisateur souhaite définir exactement les séquences de modes ou de lignes viables sans pour autant supprimer définitivement un mode : par exemple s'il ne désire prendre qu'une et une seule fois le métro s'il est précédé et suivi par du bus et dans le cas contraire utiliser le métro autant de fois qu'il le souhaite. Pour cela, une fonction d'étiquetage du graphe qui à chacun de ses arcs donne une lettre parmi un ensemble fini de lettres possibles, l'alphabet, va être utilisée. La contrainte intéressante qu'il convient d'utiliser consiste donc à imposer que le mot correspondant au plus court chemin après transposition de la séquence d'arcs du chemin en séquence de lettres de l'alphabet appartienne à un certain langage. Il a été montré que rechercher un plus court chemin dont le mot associé est contraint à être dans un langage régulier ou algébrique est un problème polynomial.

Algorithme DRegLC Pour un langage régulier, l'algorithme DRegLC ([21], [20]) est utilisé pour résoudre le plus court chemin. Il consiste tout simplement à appliquer l'algorithme de Dijkstra sur le graphe résultant du produit du graphe source et de celui correspondant à l'automate acceptant le langage en question. L'algorithme termine lorsque le nœud candidat sélectionné correspond au nœud destination dans le graphe source et que son état est un état acceptant de l'automate.
La complexité est donc polynomiale en fonction du produit du nombre de nœuds du graphe et du nombre d'états dans l'automate. Cependant, il faut noter que les chemins obtenus par l'algorithme DRegLC peuvent contenir des cycles dans le graphe initial si par exemple il est nécessaire de passer par un certain arc plusieurs fois pour obtenir

son étiquette le nombre de fois nécessaire à la satisfaction de la contrainte. Le problème consistant à déterminer des chemins sans de tels cycles sur le graphe initial sous contrainte de langage régulier est alors NP-complet. Mais heureusement, dans le cas de graphe de transport, les expressions régulières demandées dans la pratique ne tombent pas dans cette complexité.

Il est possible de coupler la contrainte de langage régulier avec la technique d'accélération ALT unidirectionnelle pour accélérer la recherche. L'algorithme SDALT [47] résultant fonctionne sur un graphe dépendent du temps et apporte des gains expérimentaux d'un ordre de grandeur dans le meilleur des cas.

2.3 Itinéraires synchronisés

Une partie des extensions possible du calcul de plus courts chemins consiste à déterminer des itinéraires dit «synchronisés», c'est à dire un ensemble d'itinéraires qu'il convient de calculer en même temps parce qu'ils intéragissent entre eux. Ces extensions sont nécessaires lorsque plusieurs acteurs entrent dans le problème d'optimisation ou lorsqu'un même usager réalisent différents trajets qu'il souhaite considérer de manière globale. C'est le cas, pour des applications de covoiturage où chaque usager à son propre itinéraire et cherche à partager une partie de son itinéraire avec d'autres usagers. Cela se rencontre également pour des problèmes d'aller et de retour d'un même usager entre deux points lorsque les deux trajets doivent passer par un même lieu. Des applications peuvent également se rencontrer dans d'autres domaines comme la robotique, par la collaboration de plusieurs robots, ou comme les réseaux sociaux pour établir des lieux de rencontres ou des itinéraires communs. Dans le contexte de la mobilité, l'objectif est de réduire l'impact des déplacements en mutualisant une partie du système de transport [14] et de faciliter l'accès à ces nouveaux services de déplacements.

2.3.1 Problèmes Aller-Retour

La problématique du calcul d'un trajet-aller-retour pour un usager a été initialement introduite dans [27], il est dénommé 2Way-MMSPP (2Way Multimodal Shortest Path Problem) dans [43]. Le problème considéré consiste à déterminer un trajet de type domicile-travail utilisant à la fois un véhicule personnel et les transports en commun. A l'aller l'utilisateur commence son trajet avec son véhicule personnel qu'il peut laisser dans un parking avant de rejoindre sa destination. Cependant, il faut pouvoir garantir qu'il puisse retrouver sa voiture le soir lors du trajet retour. L'objectif est de minimiser la somme des temps de trajet pour l'ensemble des déplacements de la journée. Dans les travaux de [27], l'approche proposée se base sur une énumération des points potentiels de parking, et quatre calculs de plus courts chemins à partir de ces différents points. Pour des raisons d'efficacité en termes de temps de calcul, l'ensemble des points ne peut

être énuméré et la méthode ne permet donc pas de garantir l'obtention de la solution optimale.

Dans les travaux de [43], l'approche proposée se base sur deux algorithme bidirectionnels, depuis l'origine et la destination. Lorsque les quatre algorithmes marquent un même sommet, un point potentiel de parking est obtenu et son coût est fourni par la somme des couts provenant des quatre algorithmes. Cependant les horaires dépendant du temps sur la partie du trajet en transport en commun nécessitent une attention particulière car le coût du trajet évalué n'est qu'une borne inférieure du coût réel. Il est alors nécessaire de ré-évaluer certaines parties du calcul Afin d'améliorer l'efficacité de la méthode proposée, des conditions d'arrêt dédiées pour ces quatres algorithmes et des calculs de bornes ont été introduits. Cette approche permet de garantir l'obtention de la solution optimale dans des temps de calculs limités à quelques secondes sur le réseau routier de la région parisienne.

Ce problème de trajet aller-retour a également été considéré en le couplant avec du covoiturage [15]. L'objectif est de déterminer un trajet aller et un trajet retour en utilisant le covoiturage. Dans ce problème, les transports en commun ne sont pas utilisés. Lors du trajet aller, le conducteur et le second usager se synchronisent en un lieu du réseau (potentiellement un parking où le second usager peut déposer son véhicule) et l'usager est amené à destination, par exemple son lieu de travail. Lors du trajet retour, un autre conducteur prend en charge l'usager piéton directement sur son lieu de travail pour le ramener au point de parking. L'approche proposée est dynamique. Elle s'appuie sur la mémorisation des trajets des différents conducteurs lorsqu'ils entrent dans le système et quand un usager souhaite être pris en charge en covoiturage, la méthode parcourt les données mémorisées pour déterminer le meilleur conducteur en intégrant des contraintes limitant les détours des usagers. L'objectif est de minimiser la somme des temps de trajets aller et retour.

2.3.2 Problèmes de covoiturage dynamique

Dans le cadre du covoiturage dynamique Bit-Monnot [26] s'est intéressé à un problème de synchronisation de deux usagers en deux points, le premier étant le lieu de rendez-vous d'où est initié un trajet commun se terminant sur le second, un point de séparation, afin que les utilisateurs terminent chacun de leur côté leur trajet. L'objectif est de déterminer les points de synchronisation permettant de minimiser les horaires d'arrivée des deux usagers. Cet objectif de minimisation est exprimé comme la somme des temps de trajets. Il est supposé que l'usager piéton peut utiliser les transports en commun pour rejoindre le point de rendez-vous et depuis le point de séparation jusqu'à sa destination. Les temps de trajets de cet usager sont alors dépendant des horaires des transports utilisés. Dans [26], le problème, appelé *2SPSPP* («2 Synchronization Points Shortest Path Problem», Problème de Plus Court Chemin Synchronisé en 2 Points), a été montré comme étant de complexité polynomiale et sa résolution se base sur cinq algorithmes de

plus courts chemins. Un sous problème spécifique a été mis en évidence, il s'agit d'un problème appelé *Problème de Meilleure Origine* («Best Origin Problem») consistant à déterminer, parmi un ensemble de sommets, celui correspondant à la meilleure origine pour atteindre une destination. Dans le contexte de graphe dépendant du temps, la résolution de ce problème nécessite des adaptations spécifiques (par rapport à l'algorithme de Dijkstra) utilisant plusieurs labels par sommet et des règles de dominance pour élaguer l'exploration. Une variante consistant à limiter le temps de trajet du piéton avant sa prise en charge par un conducteur a également été étudiée, sous l'hypothèse que si le piéton n'est pas pris en charge assez rapidement, il utilisera son véhicule personnel. Dans ce cadre, des adaptations ont été fournies permettant d'accélérer la recherche de points de synchronisation. En pratique, plus la zone de prise en charge considérée par le piéton est restreinte, plus les temps de calcul de la méthode seront efficaces. Les temps de calcul moyen obtenus dans les expérimentations effectués sur le graphe représentant la région Aquitaine et la région Midi-Pyrénées (de l'ordre de 600 000 sommets de 5 millions d'arcs) varient entre 5 secondes et 200 ms selon les restrictions imposées ou non au piéton et selon les dominances utilisées pour la résolution du problème de Meilleure Origine.

Un problème similaire de covoiturage dans un contexte de transport multimodal a également été présenté dans [16] et pour lequel il s'agit de déterminer des points intermédiaires de synchronisation entre un piéton et un conducteur. L'objectif est d'assurer au piéton une arrivée à destination plus rapide en utilisant le covoiturage qu'en utilisant les transports en commun ou la marche à pieds tout en respectant des contraintes du conducteur à la fois sur son temps de détour et sur le temps d'attente au lieu de prise en charge. Le contexte est celui d'un système impliquant un ensemble d'usagers (piétons et conducteurs). A chaque arrivée d'un nouvel usager dans le système, une proposition de covoiturage est calculée. Deux cas spécifiques sont abordés, soit le calcul d'un covoiturage est effectué uniquement lors de l'arrivée d'une demande d'un piéton, sur la base des trajets enregistrés des conducteurs, soit le calcul d'un covoiturage est effectué quel que soit l'utilisateur entrant dans le système. Pour déterminer la solution de covoiturage, il est supposé que le piéton ne peut être pris en charge par un conducteur que sur certains points de son trajet en transport en commun. Le principe de résolution se base sur le pré-calcul du trajet du piéton puis sur la recherche de chemins de substitution pour réaliser une partie du trajet en covoiturage. La recherche de chemins de substitution intègre les contraintes spécifiques du conducteur, notamment sur le détour maximal qu'il accepte de réaliser. Les approches développées sont des approches heuristiques basées sur l'utilisation de différents algorithmes de Dijkstra. Les expérimentations se sont basées sur le graphe de la région Loraine (environ 700 000 sommets et 2 millions d'arcs) et considèrent des données réelles d'offres et de demandes de covoiturage (environ 500 offres et 500 demandes). Pour un usager donné, les temps de calcul de la détermination de la meilleure offre de covoiturage varient de 3 à 6 secondes selon l'approche considérée

et la quantité d'information pré-calculée.

2.4 Chemins alternatifs

Il arrive que le plus court chemin entre deux nœuds d'un graphe ne constitue pas l'unique information qui puisse être demandée. Lorsqu'il est préférable de laisser une certaine latitude concernant le chemin à utiliser pour aller d'un nœud à un autre il faut calculer des alternatives au plus court chemin. Bien sûr, ce chemin particulier, parce qu'il est le plus court, doit être compris dans l'ensemble des chemins résultats parmi lesquels l'utilisateur devra arrêter une décision.
Le critère principal du calcul d'alternatives au plus court chemin est le nombre de solutions souhaitées mais les différents algorithmes de la littérature utilisés pour cela nécessitent également une notion de différence entre deux chemins. Cette caractérisation de chemins différents est plus ou moins implicitement définie suivant que le filtrage qu'elle impose se trouve intégré dans l'algorithme par l'utilisation de règles ou par l'objectif de la méthode considérée.

2.4.1 Méthode multi-objectifs

De la même façon que la multiplication de calculs variant les contraintes, l'ajout d'objectifs permet d'obtenir des ensembles de solutions offrant un choix varié de chemins et à chaque objectif correspond une charactéristique d'un chemin solution. Pour cela est donc définie une règle de dominance qui permet de comparer les chemins. Bien qu'une règle de dominance spécifique puisse être trouvée dans certains cas spécifiques **??**, la définition générale de la dominance forte de Pareto est la suivante : une étiquette domine une autre si et seulement si elle la domine sur chacun des objectifs. Le résultat obtenu lors d'un calcul de plus courts chemins multi-objectifs est un sous ensemble des solutions non dominées, c'est à dire telles que tout chemin de cet ensemble n'est pas meilleur que tous les autres sur l'ensemble des objectifs.

Algorithme de Martins L'algorithme de Martins [51] est une version étendue de l'algorithme de Dijkstra dans laquelle chaque nœud possède un ensemble d'étiquettes non dominées. L'exécution d'un tel algorithme donne un ensemble d'étiquettes résultat formant le front de Pareto ou ensemble des solutions non dominées. Il n'y a donc pas de plus court chemin mais une multiplicité de possibilités, chacune d'entre elle étant intéressante par rapport aux autres sur au moins un objectif.
La sélection d'un nombre restreint de solutions dans le front de Pareto constitue donc une méthode pour faire un choix ([40]) basé sur des critères de recherche ou bien pour déterminer un ensemble d'alternatives au plus court chemin.

Algorithmes bi-objectifs Parmi tous les problèmes multi-objectif, la bi-ojectivité a une place particulière en ce qu'elle permet d'obtenir un ensemble de solutions non dominées mais sur des objectifs généralement contradictoires. L'exploitation des solutions non dominées dans un contexte bi-objectif reste concevable pour un humain.
De façon particulière, le calcul de plus courts chemins bi-objectifs ayant pour objectifs la minimisation du coût d'un chemin et la minimisation du nombre de changements de modes est un problème qui est bien étudié et pour lequel il existe plusieurs algorithmes distincts ([50], [18], [33]) dont l'efficacité varie. Parmis eux, les algorithmes TLS [50] et MQLS [18] peuvent s'appliquer à la minimisation d'un entier qui est croissant le long de tout chemin comme second objectif ; cas plus général qui inclut le nombre de changement de modes. Il est également possible de considérer des aspects liés à la sécurité de circulation de vélos par l'utilisation de voies spécifiques au détriment du temps de parcours([60]), l'arbitrage en étant laissé à l'utilisateur. Enfin, comme sur les réseaux de transport il s'établit une concurrence entre la voiture et les transports en communs qui n'est pas toujours modélisée de façon fiable puisque le temps de parcours de la voiture peut être statique, il est possible de considérer une approche bi-objectif cherchant à minimiser l'utilisation de la voiture et du piéton ([22]).
Ces algorithmes peuvent donc constituer une base pour le calcul de chemins alternatifs présentant un panel de chemins favorisant plus ou moins la vitesse de déplacement ou un nombre limité de changements de modes de transport, de ruptures de charge.

2.4.2 Méthode du point de passage

Dans [13] le problème permettant de trouver des plus courts chemin alternatifs est vu comme un problème d'optimisation sous contraintes. Une fois le plus court chemin obtenu, l'algorithme doit chercher le meilleur chemin différent du plus court vérifiant les propriétés suivantes qui dépendent de trois paramètres :

- *limitation du partage («limited sharing»)* : la longueur partagée entre le chemin et le plus court doit être plus petite qu'une constante fois la longueur du plus court ;
- *optimalité locale («local optimality»)* : pour une longueur fixée en fonction de la longueur du plus court chemin, tout sous chemin du chemin à trouver doit être un plus court chemin si sa longueur est inférieure à la constante ou s'il est supérieur et que sa longueur en supprimant le premier et le dernier arc est inférieure à la constante ;
- *élasticitée limité uniforme («uniformly bounded stretch»)* : tout sous chemin du chemin recherché doit être inférieur, à un facteur fixé prêt, au plus court entre son premier et dernier nœud.

Même en restreignant avec ces critères et des constantes bien choisies, le nombre de chemins potentiels est trop important pour être traité par un humain. Une façon de limiter ces possibilités de façon drastique est de se limiter à sélectionner les plus courts

chemins passant par un nœud et qui restent admissibles au regard des critères précédents. Avec cette méthode utilisant un point de passage obligé sur le chemin entre l'origine et la destination, le temps de calcul revient à lancer deux algorithmes de Dijkstra, l'un depuis l'origine et l'autre depuis la destination, en laissant l'exploration visiter l'ensemble du graphe. De plus, les chemins générés ont de bonnes propriétés vis à vis des propriétés à respecter pour trouver une solution admissible puisque les chemins passant par un point de passage minimisent l'élasticité, et puisqu'ils sont formés de deux plus courts chemins l'optimalité locale ne peut être violée qu'autour du point de passage.

2.4.3 Méthode des plateaux

Un plateau ([19], [55]) est un chemin simple constitué d'une partie sur l'arbre de plus courts chemins avant depuis l'origine et d'une autre sur l'arbre de plus courts chemins arrière depuis la destination tel que au moins deux de ses nœuds consécutifs aient une somme identique sur les deux arbres. Comme le nombre de plateaux est très grand, un rang est attribué à chacun d'eux en soustrayant le coût le long des nœuds qui ont permis de constituer ce plateau au coût du chemin entier. De cette façon, un tri sur les plateaux puis sur les coûts donne les meilleures alternatives et par coût croissant en cas d'égalité. La méthode des plateaux génère des solutions qui peuvent faire un détour très important dès que les graphes deviennent grands car il existe alors des plateaux dont la longueur leur confère un rang très haut mais qui effectuent alors un détour qui devient inacceptable pour un usager.

2.4.4 Méthode des pénalités

La méthode dite des «pénalités» ([28], [19]) consiste à calculer des plus courts chemins successivement sur un graphe dont les coûts sont modifiés en fonction des résultats obtenus dans les étapes précédentes. Initialement, un plus court chemin sur le graphe originel est obtenu. Puis, itérativement pour tous les chemins à obtenir, les coûts des arcs du graphe sont modifiés en fonction des chemins trouvés jusqu'à présent et un plus court chemin est calculé. Ensuite, en fonction de la méthode, ce chemin peut être directement intégré à la solution, ou bien, il peut d'abord faire l'objet d'une analyse afin de déterminer s'il s'agit d'une bonne alternative auquel cas il est ajouté à la solution, sinon le graphe est modifié à nouveau pour une nouvelle itération sans que le chemin fasse partie des solutions.

Toute la subtilité de la méthode repose donc sur les choix effectués pour modifier le coût des arcs et une première possibilité est d'augmenter le coût par une constante pour chacun des arcs des chemins. Cependant, cette opération a pour effet de sur-représenter les chemins constitués de nombreux arcs, la multiplication du coût par un facteur sera donc plus appropriée. Plus ce facteur sera important et plus les nouveaux chemins dévieront des précédents.

Toutefois, cette méthode peut donner de nombreux chemins comportant uniquement de

petits détours le long des chemins déjà solution. Pour contourner cette limitation, le coût des arcs se trouvant autour de chemins, idéalement adjacents, sont également modifiés afin d'éliminer les détours contenus dans un «tube» autour des chemins.

2.4.5 Méthode des K plus courts chemins

Le calcul de k plus courts chemins consiste à énumérer les chemins entre une origine et une destination par coûts croissants. Il se décline sous deux formes. La première trouve des chemins contenant des cycles([35], [36]), possiblement sur l'origine et la destination en fonction des variantes, alors que la seconde se limite à lister une partie de l'ensemble fini des chemins sans cycles ([64]).
Pour déterminer des alternatives il est donc possible de se baser sur cet ensemble de chemins afin d'en extraire des alternatives. Cela peut être fait de façon incrémentale en partant du plus court chemin comme ensemble solution puis en déterminant pour chaque nouveau chemin s'il constitue une bonne alternative étant donné les autres chemins déjà choisis afin de l'ajouter à l'ensemble des solutions ou de le rejeter. Cette façon de procéder observe un horizon (ensemble de chemins) qui est limité à l'ensemble courant des solutions et au chemin suivant alors qu'une approche plus globale pourrait selectionner un sous ensemble de l'ensemble des chemins possibles suivant un ou des objectifs, au détriment de la complexité qui en toute généralité devient alors exponentielle.

Chapitre 3

K plus courts chemins

Le calcul des k plus courts chemins, avec cycles ou élémentaires, a été étudié dans la littérature et les principales méthodes de résolution sont présentées ci-après. Suivant la nature du problème (recherche de chemins sans cycle ou de chemins élémentaires), la complexité et les performances expérimentales varient. D'après les complexités théoriques, il existe une différence au moins linéaire entre les deux familles de méthodes. Le temps de calcul devient donc non négligeable lorsque l'on considère des graphes de grande taille.
Pour l'obtention de chemin sans cycle, la concurrence entre des méthodes calculant des chemins avec cycles couplées à des techniques de filtrage pour obtenir des chemins élémentaires et des méthodes calculant directement ces chemins n'a pas été étudiée. Toutefois, le nombre de chemins avec cycles compris (en terme de coût) entre deux chemins élémentaires étant factoriel, il n'est pas évident qu'une telle concurrence existe.
Un des objectifs de ce travail est de déterminer si les méthodes de recherche des k plus courts chemins avec cycles peuvent s'adapter efficacement pour déterminer k plus courts chemins élémentaires.

D'autre part, rappelons que cette thèse s'inscrit dans le cadre de graphes spécifiques, à savoir des réseaux de transport multimodaux et dépendant du temps dans lesquels on cherche à obtenir des chemins contraints à un langage régulier. Vu qu'il n'existait, à notre connaissance, pas de travaux visant à déterminer des k plus courts chemins dans un tel contexte, un de nos objectifs a été d'adapter des méthodes existantes pour des graphes multimodaux et dépendant du temps ou de proposer de nouvelles méthodes de recherche de k plus courts chemins dans ce cadre.

Après avoir introduit les principes généraux de calcul de k plus courts chemins (section 3.1), nous présentons les adaptations des algorithmes de la littérature aux réseaux de transport dans la section 3.2. En section 3.3.1, nous introduisons un nouvel algorithme de cacul de plus court chemin avec cycle, appelé IEA, puis nous proposons une procédure interne permettant de couper les cycles 3.4. Une étude expérimentale permettant de comparer ces différentes méthodes est présentée dans la section 3.5. Enfin, des pistes d'exploitation de l'ensemble des chemins sans cycles obtenus afin de déterminer des trajets alternatifs pertinents sont exposées dans la section 3.6.

3.1 Principes généraux des méthodes de k plus courts chemins

Le problème des k plus courts chemins correspond dans la littérature à deux familles de problèmes selon que les chemins calculés soient élémentaires ou avec cycles. Plusieurs de ces méthodes nécessitent comme sous routine à un moment donné, une procédure calculant un plus court chemin élémentaire entre deux nœuds, la fonction PlusCourtChemin présentée dans le pseudo-code de l'algorithme 1 est supposée connue que ce soit un algorithme de Dijkstra ou bien une version plus évoluée mais ayant toujours la même complexité temporelle. Cette procédure considère un graphe $G = (V, E)$ modélisant un réseau de transport multimodal et dépendant du temps, une origine o ayant un coût initial c_0, une destination d et une fonction f de valuation des arcs. Elle retourne la valeur c du plus court chemin origine-destination ainsi que l'ensemble des sommets constituant ce plus court chemin.

Algorithme 1 Algorithme arbitraire de calcul de plus court chemin

1: **function** PlusCourtChemin($G = (V, E)$, $o \in V$, $d \in V$, $f : E \to \mathbb{R}_+$, c_0) :($c \in \mathbb{R}, (x_i)_{x_i \in V}$)
2: $P \leftarrow$ le plus court chemin sur G entre o et d suivant la fonction d'évaluation f et un coût initial c_0
3: **retourner** (f(P), P)
4: **Fin function**

Dans les algorithmes de calcul de plusieurs plus courts chemins, la notion récurrente est celle de calcul de déviations. A partir de sous-chemins déjà solutions, l'obtention du chemin suivant se fait par l'exploration de déviations sur les chemins existants pour obtenir de nouveaux chemins avec des coûts plus grands.

3.1.1 Calcul des k plus courts chemins élémentaires

L'ensemble des chemins élémentaires sur un graphe a une cardinalité dénombrable et donc les rechercher consiste à les énumérer dans leur ordre croissant de coût.

Algorithme de Yen L'algorithme de Yen [64] calcule incrémentalement de tels chemins. A chaque itération, un nouveau plus court chemin est obtenu en appliquant l'algorithme 1 de calcul de plus court chemin sur un graphe modifié en fonction des chemins solutions déjà énumérés jusqu'à présent.

Le pseudo code de l'algorithme 2 décrit de façon effective le fonctionnement de l'algorithme de Yen. Tout d'abord l'ensemble des chemins candidats H est initialisé avec le plus court chemin et son coût entre l'origine o et la destination d (ligne 2). Dans le cas où cet ensemble serait vide, par exemple lorsque tous les chemins élémentaires ont été énumérés, où lorsque le nombre souhaité de solutions est atteint, alors l'algorithme termine (ligne 3). Dans les autres scenarios, le meilleur de ces candidats est extrait de

l'ensemble H et ajouté comme solution dans le tableau S des solutions ordonnées par coût (lignes 4 à 6).
Ensuite, pour calculer le $k+1^{ième}$ chemin en connaissant les $k^{ièmes}$ précédents, l'algorithme procède de la façon suivante : pour chaque sommet x_i du chemin $(x_1, ..., x_n)$ obtenu à l'itération précédente, un nouveau graphe $G' = (V', E')$ est contruit à partir du graphe initial $G = (V, E)$ et des chemins déjà calculés et présents dans S. Initialement ce nouveau graphe est identique au graphe G (ligne 9). Puis les arcs sortant du nœud x_i et appartenant à un chemin déjà solution dont la racine jusqu'au nœud considéré est identique à celle du chemin précédent sont supprimés (lignes 10 à 14). De même, tous les $n-1$ nœuds précédent x_i dans le chemin deviennent inaccessibles ce qui entraîne aussi la suppression des arcs qui leurs sont adjacents (lignes 15 à 18).
Le coût du chemin précédent s'arrêtant au nœud x_i est obtenu par l'appel de la fonction f (ligne 19) sur chaque arc le composant. Il est ensuite utilisé pour calculer le plus court chemin s'il existe sur le graphe modifié G' entre les nœuds x_i et d (ligne 20). Il est ajouté à l'ensemble des solutions potentielles dans le cas où il n'y figurait pas déjà (ligne 21).

Algorithme 2 Algorithme de Yen

```
1: function YEN(G = (V, E), o ∈ V, d ∈ V, f : E → ℝ+, c0 ∈ ℝ) :[(c ∈ ℝ, (xi)_{xi∈V})]
2:     H ← {PLUSCOURTCHEMIN(G, o, d, f, c0)}
3:     Tant que H ≠ ∅ ∧ |S| ≠ k faire
4:         s ← min_c(H = {(c, P)})
5:         S[|S| + 1] ← s
6:         H ← H \ {s}
7:         (−, (x1, ..., xn)) ← ITÈRE(S)
8:         Pour tout xi ∈ (x1, ..., x_{n−1}) faire
9:             (V', E') ← G
10:            Pour tout (−, (y1, ..., ym)) ∈ S faire
11:                Si (x1, ..., xi) = (y1, ..., yi) alors
12:                    E' ← E' \ {(yi, y_{i+1})}
13:                Fin Si
14:            Fin Pour
15:            Pour tout x ∈ (x1, ..., x_{i−1}) faire
16:                V' ← V' \ {x}
17:                E' ← E' \ {(y, x) ∈ E, (x, y) ∈ E | y ∈ V}
18:            Fin Pour
19:            c ← Σ_{(x,y)∈((x1,x2),...,(x_{i−1},xi))} f((x, y))
20:            (c, (y1, ..., ym)) ← PLUSCOURTCHEMIN((V', E'), xi, d, f, c)
21:            H ← H ∪ {(c, ((x1, ..., x_{i−1}), (y1, ..., ym)))}
22:        Fin Pour
23:    Fin Tant que
24:    retourner S
25: Fin function

1: function ITÈRE(S = [(xi)]) :(xj) ∈ 𝒫((xi))
2:     retourner S[|S| − 1]
3: Fin function
```

L'extension de Lawler [49] permet d'éviter le calcul de doublons en changeant la fonction ITÈRE qui par défaut ne retourne que le dernier chemin solution. En effet, lorsqu'un

chemin possède un préfixe commun avec un chemin calculé juste avant, l'extension de chacun des nœuds de cette partie identique calcule un chemin déjà déterminé à l'étape précédente. Pour cela, la nouvelle fonction ITÈRE donnée dans le pseudo-code 3 analyse à quel nœud le chemin précédemment trouvé (ligne 5) diverge (ligne 8) de l'antépénultième (ligne 6) et ne retourne que la fin de ce chemin en conservant le premier nœud commun (ligne 11). Ensuite l'algorithme de Yen reprendra la main et calculera usuellement les déviations sur les graphes modifiés successivement sur chacun des nœuds restants du dernier chemin solution.

Algorithme 3 Extension de Lawler

```
1: function ITÈRE(S)S = [(x_i)] :(x_j) ∈ 𝒫((x_i))
2:     Si |S| = 1 alors
3:         retourner S[0]
4:     Fin Si
5:     (−, (x_1, ..., x_n)) ← S[|S| − 1]
6:     (−, (y_1, ..., y_m)) ← S[|S| − 2]
7:     i ← 0
8:     Tant que i < n ∧ i < m ∧ x_{i+1} = y_{i+1} faire
9:         i ← i + 1
10:    Fin Tant que
11:    retourner (x_i, ..., x_n)
12: Fin function
```

Exemple La partie suivante détaille l'application de l'algorithme de Yen pour trouver l'ensemble des chemins élémentaires entre x_1 et x_2 sur le graphe G_1 présenté en figure 3.1.

A l'étape zéro, l'algorithme est initialisé avec le plus court chemin (x_1, x_2). Par la suite, et pour chaque étape, est indiquée :

— l'exploration à partir du chemin solution courant,

— la solution locale si elle existe pour une déviation particulière,

— et l'ensemble des chemins candidats.

Lorsqu'un chemin est inséré dans la liste des candidats il est précédé du marqueur $\bullet$ alors que s'il était pré-existant le marqueur est alors $\times$. Le coût de chaque chemin est indiqué entre crochets à leur fin. Le nœud souligné correspond au nœud sur lequel les déviations sont calculées et les ensembles E' et V' figurent le graphe modifié sur lequel elles sont calculées.

Sur cet exemple, à partir de l'étape 1 la première itération se fait sur le nœud x_1 pour lequel un nouveau chemin est obtenu. A l'étape 2, il est possible de voir les itérations internes sur chacun des nœuds du chemin sauf le dernier.

A chaque fois, le graphe est modifié en fonction des chemins résultats déjà obtenus. Lors de l'étape 3, le chemin $(x_1, x_4, x_5, x_3, x_2)$ est de nouveau obtenu et l'étape suivante notée $\star$ montre l'endroit précis où l'algorithme étendu par la version de Lawler aurait commencé. Puisque les chemins π^2 et π^3 possèdent le même préfixe (x_1, x_3), l'extension aurait directement commencé au dernier nœud de la racine commune à savoir x_3. Une itération aurait alors été gagnée ici, mais dans de plus grands graphes cette amélioration

peut être bien plus conséquente.

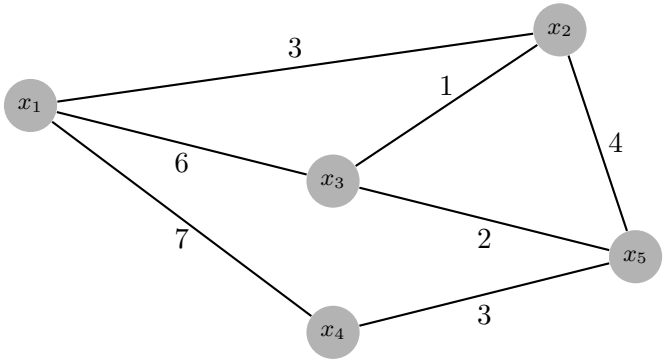


FIGURE 3.1 – Graphe exemple G_1

3.1.2 Calcul des k plus courts chemins avec cycles

Dans de nombreuses applications, le calcul de k plus courts chemins élementaires est intuitivement souhaitable, c'est le cas en particulier pour les applications de mobilité. Cependant, dans certains problèmes, des chemins comportant des cycles représentent une information importante, typiquement pour calculer des chemins de secours. C'est le cas considéré par les algorithmes d'Eppstein et REA présentés ci-après.

Algorithme d'Eppstein L'algorithme de Eppstein [36] cherche des déviations successives en se basant sur un graphe extrait du graphe initial. Dans un premier temps, un arbre de plus court chemin en parcours arrière depuis la destination est calculé. A partir de cet arbre, il est possible d'en déduire le complémentaire du graphe initial, c'est à dire tous les arcs du graphe n'appartenant pas à l'arbre de plus court chemin. Ces déviations sont ensuite triées à l'aide de tas sur chacun des nœuds du graphe puis sont aggrégées en une structure d'arbre les combinant dans sa profondeur. Le calcul des k plus courts chemins repose donc finalement sur un parcours en largueur de cet arbre qui énumère par coût croissant l'ensemble des déviations successives à appliquer pour obtenir de nouveaux chemins déviant de l'arbre de plus court chemin. Une méthode particulière de sélection de k éléments dans un tas [38] permet d'obtenir une représentation implicite des chemins solutions ce qui conduit finalement à une complexité au pire de $O(|E| + |V| \times log(|V|) + k)$. Cette complexité théorique est la meilleure complexité obtenue pour le calcul des k plus courts chemins contenant des cycles.
Une implémentation particulière [46] permet d'améliorer les performances pratiques de cet algorithme sans en changer la complexité théorique. Pour cela, l'arbre de déviations n'est pas calculé directement mais obtenu à la volée de façon paresseuse, c'est à dire que que son évaluation n'est effectuée qu'en dernier recours lorsqu'elle est nécessaire avant de manquer un optimum local.
Une dernière implémentation appelée K^* [17] réutilise ce principe pour calculer k plus

Etape	Exploration	Solution locale	Candidats
0	$(\underline{x_1})[0]$	$(x_1, x_2)[3]$	$\surd(x_1, x_2)[3]$
1	$\pi^1 = (x_1, x_2)[3]$		
	$(\underline{x_1}, x_2)[0]$ $E' \leftarrow E \setminus \{(x_1, x_2)\}$ $V' \leftarrow V \setminus \emptyset$	$(x_1, x_3, x_2)[7]$	$\surd(x_1, x_3, x_2)[7]$
2	$\pi^2 = (x_1, x_3, x_2)[7]$		
	$(\underline{x_1}, x_3, x_2)[0]$ $E' \leftarrow E \setminus \{(x_1, x_2), (x_1, x_3)\}$ $V' \leftarrow V \setminus \emptyset$	$(x_1, x_4, x_5, x_3, x_2)[13]$	$\surd(x_1, x_4, x_5, x_3, x_2)[13]$
	$(x_1, \underline{x_3}, x_2)[6]$ $E' \leftarrow E \setminus \{(x_3, x_2)\}$ $V' \leftarrow V \setminus \{x_1\}$	$(x_3, x_5, x_2)[6]$	$\surd(x_1, x_3, x_5, x_2)[12]$ $(x_1, x_4, x_5, x_3, x_2)[13]$
3	$\pi^3 = (x_1, x_3, x_5, x_2)[12]$		
	$(\underline{x_1}, x_3, x_5, x_2)[0]$ $E' \leftarrow E \setminus \{(x_1, x_2), (x_1, x_3)\}$ $V' \leftarrow V \setminus \emptyset$	$(x_1, x_4, x_5, x_3, x_2)[13]$	$\times(x_1, x_4, x_5, x_3, x_2)[13]$
	$\star(x_1, \underline{x_3}, x_5, x_2)[6]$ $E' \leftarrow E \setminus \{(x_3, x_2), (x_3, x_5)\}$ $V' \leftarrow V \setminus \{x_1\}$	$\emptyset$	$(x_1, x_4, x_5, x_3, x_2)[13]$
	$(x_1, x_3, \underline{x_5}, x_2)[8]$ $E' \leftarrow E \setminus \{(x_5, x_2)\}$ $V' \leftarrow V \setminus \{x_1, x_3\}$	$\emptyset$	$(x_1, x_4, x_5, x_3, x_2)[13]$
4	$\pi^4 = (x_1, x_4, x_5, x_3, x_2)[13]$		
	$(\underline{x_1}, x_4, x_5, x_3, x_2)[0]$ $E' \leftarrow E \setminus \{(x_1, x_2), (x_1, x_3), (x_1, x_4)\}$ $V' \leftarrow V \setminus \emptyset$	$\emptyset$	$\emptyset$
	$(x_1, \underline{x_4}, x_5, x_3, x_2)[7]$ $E' \leftarrow E \setminus \{(x_4, x_5)\}$ $V' \leftarrow V \setminus \{x_1\}$	$\emptyset$	$\emptyset$
	$(x_1, x_4, \underline{x_5}, x_3, x_2)[10]$ $E' \leftarrow E \setminus \{(x_5, x_3)\}$ $V' \leftarrow V \setminus \{x_1, x_4\}$	$(x_5, x_2)[4]$	$\surd(x_1, x_4, x_5, x_2)[14]$
	$(x_1, x_4, x_5, \underline{x_3}, x_2)[12]$ $E' \leftarrow E \setminus \{(x_3, x_2)\}$ $V' \leftarrow V \setminus \{x_1, x_4, x_5\}$	$\emptyset$	$(x_1, x_4, x_5, x_2)[14]$
5	$\pi^5 = (x_1, x_4, x_5, x_2)[14]$		
	$(\underline{x_1}, x_4, x_5, x_2)[0]$ $E' \leftarrow E \setminus \{(x_1, x_2), (x_1, x_3), (x_1, x_4)\}$ $V' \leftarrow V \setminus \emptyset$	$\emptyset$	$\emptyset$
	$(x_1, \underline{x_4}, x_5, x_2)[7]$ $E' \leftarrow E \setminus \{(x_4, x_5)\}$ $V' \leftarrow V \setminus \{x_1\}$	$\emptyset$	$\emptyset$
	$(x_1, x_4, \underline{x_5}, x_2)[10]$ $E' \leftarrow E \setminus \{(x_5, x_2), (x_5, x_3)\}$ $V' \leftarrow V \setminus \{x_1, x_4\}$	$\emptyset$	$\emptyset$

TABLE 3.1 – Application de l'algorithme de Yen au graphe de la figure 3.4.

courts chemins sur des graphes dont la connaissance n'est que partielle, soit parce qu'ils sont trop volumineux, soit parce qu'ils sont découverts au fur et à mesure de l'exploration. Cet algorithme ne se base plus sur un arbre de plus courts chemins en parcours arrière depuis la destination puisqu'il n'est pas possible de l'obtenir en raison de la connaissance partielle du graphe mais exécute alternativement la construction d'un arbre de plus courts chemins en parcours avant et la construction de l'arbre des déviations dès que cela est nécessaire. Cette méthode permet de plus d'introduire l'utilisation d'heuristiques pour s'exécuter tout comme l'algorithme A^*. Ses performances expérimentales sont donc bien meilleures que celles des deux algorithmes susmentionés.

Algorithme REA L'algorithme REA [45] (Recursive Enumeration Algorithm - Algorithme d'énumération récursive), de la même façon que l'algorithme de Eppstein s'appuie sur le calcul d'un arbre de plus courts chemins pour ensuite calculer les chemins par des appels récursifs depuis la destination et utilisant cette structure initiale. Le pseudo code 4 détaille la fonction principale d'initialisation et de contrôle REA et sa fonction interne d'appels récursifs CheminSuivant.
Les variables π_v^k correspondent au $k^{ième}$ chemin allant de l'origine o au nœud v. Elle est constituée d'un triplet $(c, P = (x_0, ..., x_i, ..., v), (u, k'))$ indiquant :

- le coût c du chemin ;
- P son chemin depuis l'origine ;
- son chemin prédécesseur $\pi_v^{k'}$ donnée par le couple (u, k').

Tout d'abord, l'algorithme calcule (à la ligne 3) un arbre de plus court chemins en évaluation avant de l'origine o vers tous les autres nœuds du graphe pour initialiser la structure résultat pour k égal à 1. Pour les autres valeurs de k, les résultats sont vides et des tas locaux H_v de candidats potentiels en chacun des nœuds sont initialisés. Puis, les tas locaux de chacun des nœuds sont initialisés avec les candidats potentiels au deuxième chemin. Pour chaque nœud v, il faut déterminer pour chacun de ses prédécesseurs u s'il est le nœud juste avant v dans le chemin depuis l'origine vers v (ligne 12). Si c'est le cas rien n'est fait puisqu'il s'agit du résultat trouvé pour le premier chemin. Sinon, il faut ajouter le chemin étendu, à partir de u vers v par l'arc qui les relie, à la liste des candidats potentiels pour le nœud v.
Ensuite, la boucle principale appelle la fonction CheminSuivant (ligne 20) autant de fois que nécessaire pour atteindre la valeur de k souhaitée et tant que des chemins peuvent être obtenus (ligne 18) car l'algorithme peut être appliqué sur des graphes pour lesquels il n'est pas possible d'obtenir de cycle (pour les arbres en particulier).

Algorithme 4 Algorithme REA

1: **function** REA($G = (V, E)$, $o \in V$, $d \in V$, $k \in \mathbb{N}$, $f : E \to \mathbb{R}_+$, $c_0 \in \mathbb{R}$, $l_o \in \mathbb{B}$) :res
2: **Pour tout** $v \in V$ **faire**
3: $(P, c) \leftarrow$ PlusCourtChemin(G, o, v, f, c_0)
4: $\pi_v^1 \leftarrow (c, P, \emptyset)$
5: $\pi_v^i \leftarrow \emptyset$, $\forall i \in [\![2, k]\!]$
6: $H_v \leftarrow \emptyset$
7: **Fin Pour**
8: **Pour tout** $v \in V$ **faire**
9: **Pour tout** $(u, v) \in E$ **faire**
10: $(c, (o, ..., w_i, ..., u), -) \leftarrow \pi_u^1$
11: $(-, (o, ..., w_i', ..., v), -) \leftarrow \pi_v^1$
12: **Si** $(o, ..., w_i, ..., u, v) \neq (o, ..., w_i', ..., v)$ **alors**
13: $H_v \leftarrow H_v \cup \{(c + f((u, v)), (o, ..., w_i, ..., u, v), (u, 1))\}$
14: **Fin Si**
15: **Fin Pour**
16: **Fin Pour**
17: $(k', \varepsilon) \leftarrow (1, \textbf{Vrai})$
18: **Tant que** $k' < k \wedge \varepsilon$ **faire**
19: $k' \leftarrow k' + 1$
20: $(\varepsilon, \pi, H) \leftarrow$ CheminSuivant$(G, o, d, k', \pi, H, l_o)$
21: **Fin Tant que**
22: **retourner** π
23: **Fin function**

1: **function** CheminSuivant($G = (V, E)$, $o \in V$, $v \in V$, $k \in \mathbb{N}$, $\pi \subset \mathbb{R} \times V^* \times (V \times \mathbb{N})$, $H \subset \pi$, $l_o \in \mathbb{B}$) :($existe \in \mathbb{B}, \pi' \subset \mathbb{R} \times V^* \times (V \times \mathbb{N}), H' \subset \pi'$)
2: **Si** $\neg l_o \wedge v = o$ **alors**
3: **retourner** (**Faux**, π, H)
4: **Fin Si**
5: **Si** $\neg(v = o \wedge k = 2)$ **alors**
6: $(-, -, (u, k')) \leftarrow \pi_v^{k-1}$
7: $\varepsilon \leftarrow \pi_u^{k'+1} = \emptyset$
8: **Si** ε **alors**
9: $(\varepsilon, \pi, H) \leftarrow$ CheminSuivant$(G, o, u, k' + 1, \pi, H, l_o)$
10: **Fin Si**
11: **Si** ε **alors**
12: $(c, (o, ..., w_i, ..., u), -) \leftarrow \pi_u^{k'+1}$
13: $H_v \leftarrow H_v \cup \{(c + f((u, v)), (o, ..., w_i, ..., u, v), (u, k' + 1))\}$
14: **Fin Si**
15: **Fin Si**
16: $\varepsilon \leftarrow$ **Faux**
17: **Si** $H_v \neq \emptyset$ **alors**
18: $\pi_v^k \leftarrow min(H_v)$
19: $H_v \leftarrow H_v \setminus \{\pi_v^k\}$
20: $\varepsilon \leftarrow$ **Vrai**
21: **Fin Si**
22: **retourner** (ε, π, H)
23: **Fin function**

La fonction CheminSuivant calcule récursivement les $k + 1^{ième}$ chemins en autant de nœuds que nécessaire pour la constitution du $k^{ième}$ chemin en un nœud donné du graphe. Lorsque le nœud sur lequel est appelé la fonction CheminSuivant n'est pas l'origine et que k est supérieur à 2 (ligne 5), alors il faut récupérer le prédécesseur $\pi_u^{k'}$ sur le chemin π_v^{k-1}, c'est à dire le dernier connu sur v, tel que $(\pi_u^{k'}, v) = \pi_v^{k-1}$ (ligne 6).

Il existe nécessairement sauf pour l'origine mais cette éventualité est écartée par le test effectué juste en amont. C'est ensuite sur ce nœud u qu'est appelée récursivement la procédure (ligne 9) en demandant le $k' + 1^{ième}$ chemin s'il n'existe pas. Lorsqu'il est obtenu, il est ajouté à l'ensemble de candidats du nœud v après avoir été étendu jusqu'à lui depuis u (ligne 13). A la fin, le meilleur chemin candidat est extrait (ligne 17), c'est le $k^{ième}$ chemin du nœud v.
L'algorithme présenté ci-dessus ne correspond pas tout à fait à celui décrit dans l'article originel. La différence se trouve au niveau du calcul effectué dans la fonction REA pour l'initialisation des tas en chaque nœud pour les candidats au deuxième chemin. Dans l'algorithme, y compris la version implémentée pour cette thèse, cela est fait dans la fonction CheminSuivant dans une clause vérifiant si la valeur de k' demandée est de 2. Cette différence est introduite ici dans un effort de simplification de l'algorithme pour le lecteur mais elle ne serait pas très performante en pratique notamment si le graphe est de grande taille, où dans ce cas une évaluation paresseuse des candidats au deuxième plus court chemin serait bien plus efficace.
De plus, une extension que nous proposons consiste à interdire les chemins contenant plusieurs fois l'origine, c'est à dire qui bouclent sur l'origine au moyen d'une condition permettant de ne jamais calculer π_o^k pour des valeurs de k plus grande que 1 si la valeur l_o est à **Faux** (ligne 2).

Afin de bien voir l'enchaînement des appels récursifs de cet algorithme, son exécution est expliquée sur le graphe G_2 de la figure 3.2 entre les nœuds x_1 et x_2. Les cycles à l'origine serons ignorés dans le but de limiter le nombre de chemins comportant des cycles entre deux chemins sans cycle.

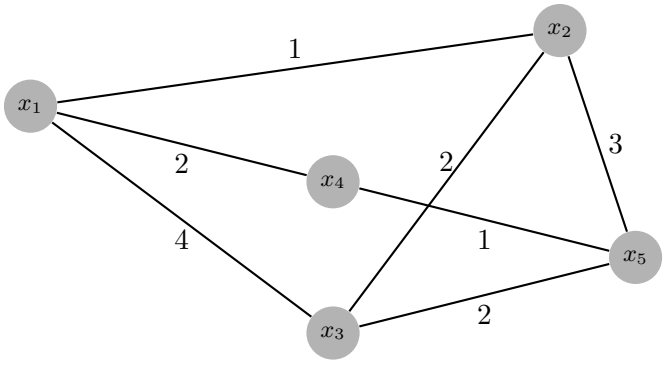


Figure 3.2 – Graphe exemple G_2

Le tableau 3.2 donne les valeurs obtenues lors du calcul du plus court chemin de x_1 vers tous les sommets du graphe pour $k = 1$. La première ligne donne le nom du chemin calculé, la seconde ligne fournit le coût de ce chemin, la troisième ligne présente le chemin sous forme d'une séquence de sommets, et la quatrième donne le nœud prédécesseur ainsi que le numéro du chemin considéré pour l'extension.

Chemin	$\pi^1_{x_1}$	$\pi^1_{x_2}$	$\pi^1_{x_3}$	$\pi^1_{x_4}$	$\pi^1_{x_5}$
Coût	0	1	3	2	3
Chemin	(x_1)	(x_1, x_2)	(x_1, x_2, x_3)	(x_1, x_4)	(x_1, x_4, x_5)
Prédécesseur	$\emptyset$	$(x_1, 1)$	$(x_2, 1)$	$(x_1, 1)$	$(x_4, 1)$
Ecriture simplifiée	$(x_1), 0$	$(\pi^1_{x_1}, x_2), 1$	$(\pi^1_{x_2}, x_3), 3$	$(\pi^1_{x_1}, x_4), 2$	$(\pi^1_{x_4}, x_5), 3$

TABLE 3.2 – Initialisation de l'algorithme REA sur le graphe de la figure 3.2

Le tableau 3.3, présente les calculs effectués pour $k = 2$. Considérons le cas du sommet x_2 qui a trois prédécesseurs : x_1, x_3 et x_5. Pour chacun de ces prédécesseurs y, l'algorithme teste si le chemin étendu de y à x_2 (π^1_y, x_2) correspond au premier chemin trouvé en x_2, c'est à dire $\pi^1_{x_2}$. Si tel est le cas, le chemin (π^1_y, x_2) est écarté, sinon il est conservé et son coût est évalué. Pour le sommet x_2, le chemin $(\pi^1_{x_1}, x_2)$ est écarté et les chemins $(\pi^1_{x_3}, x_2)$ et $(\pi^1_{x_5}, x_2)$ sont conservés avec des couts respectifs de 5 et de 6.

Nœud	x_2	x_3	x_4	x_5
Tests	$(\pi^1_{x_1}, x_2) = \pi^1_{x_2}$ $(\pi^1_{x_3}, x_2) \neq \pi^1_{x_2}$ $(\pi^1_{x_5}, x_2) \neq \pi^1_{x_2}$	$(\pi^1_{x_1}, x_3) \neq \pi^1_{x_3}$ $(\pi^1_{x_2}, x_3) = \pi^1_{x_3}$ $(\pi^1_{x_5}, x_3) \neq \pi^1_{x_3}$	$(\pi^1_{x_1}, x_4) = \pi^1_{x_4}$ $(\pi^1_{x_5}, x_4) \neq \pi^1_{x_4}$	$(\pi^1_{x_2}, x_5) \neq \pi^1_{x_5}$ $(\pi^1_{x_3}, x_5) \neq \pi^1_{x_5}$ $(\pi^1_{x_4}, x_5) = \pi^1_{x_5}$
Candidats	$(\pi^1_{x_3}, x_2), 5$ $(\pi^1_{x_5}, x_2), 6$	$(\pi^1_{x_1}, x_3), 4$ $(\pi^1_{x_5}, x_3), 5$	$(\pi^1_{x_5}, x_4), 4$	$(\pi^1_{x_2}, x_5), 4$ $(\pi^1_{x_3}, x_5), 5$

TABLE 3.3 – Initialisation des tas locaux à chaque nœud pour l'algorithme REA sur le graphe de la figure 3.2

Le tableau 3.4 rassemble les actions effectuées pour chaque appel à la fonction CHEMINSUIVANT ainsi que les ensembles de candidats associés à chaque nœud. Les appels à CHEMINSUIVANT sont suivis de la profondeur d'appel indiquée entre crochets.

Le premier appel effectué se fait sur le nœud x_2 afin d'y déterminer le deuxième chemin depuis x_1. Comme il s'agit du cas particulier où k vaut deux, il faut regarder l'ensemble des prédécesseurs de x_2 et étendre depuis leur premier chemin vers x_2 sauf pour le nœud x_1 qui est déjà le prédécesseur de $\pi^1_{x_2}$ (ce calcul a été présenté dans le tableau 3.3).

Ensuite, la fonction CHEMINSUIVANT est appelée sur le nœud x_1 mais puisque les cycles à l'origine sont interdits, l'appel se termine et une solution pour le second chemin en x_2 est déterminée à partir de l'ensemble des candidats de ce nœud. Celui de plus petit coût est alors sélectionné : $\pi^{\mathbf{2}}_{\mathbf{x_2}} = (\pi^{\mathbf{1}}_{\mathbf{x_3}}, \mathbf{x_2}) = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_2})$ de coût 5. Pour le calcul du troisième chemin en x_2 (CHEMINSUIVANT$(x_2, 3)$) l'algorithme doit dans un premier temps déterminer le deuxième chemin en x_3, $(x_3, 2)$. Ce dernier doit alors étendre le deuxième plus court chemin arrivant en x_2. Or $NextPath(x2, 2)$ a déjà été calculé et il ne reste donc plus qu'à l'étendre pour produire un nouveau candidat de coût 7 en x_3. Le deuxième plus court chemin arrivant en x_3 est alors choisit parmis la liste des candidats et il s'agit de celui de coût 4. Lors du retour de la fonction CHEMINSUIVANT$(x_3, 2)$, le nouveau chemin $\pi^2_{x_3}$ est utilisé pour produire le candidat $(\pi^2_{x_3}, x_2)$ de coût 6 sur le nœud x_2.

CheminSuivant(x,k)	Action	x_2	x_3	x_4	x_5
$(\mathbf{x_2},\mathbf{2})[\mathbf{0}]$	CheminSuivant$(x_1,2)$	$(\pi^1_{x_3},x_2),5$ $(\pi^1_{x_5},x_2),6$	$(\pi^1_{x_1},x_3),4$ $(\pi^1_{x_5},x_3),5$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
$(x_1,2)[1]$	$\times$				
$(x_2,2)[0]$	$\pi^{\mathbf{2}}_{\mathbf{x_2}} = (\pi^{\mathbf{1}}_{\mathbf{x_3}},\mathbf{x_2}) = (\mathbf{x_1},\mathbf{x_2},\mathbf{x_3},\mathbf{x_2})$, Coût 5				
$(\mathbf{x_2},\mathbf{3})[\mathbf{0}]$	CheminSuivant$(x_3,2)$	$(\pi^1_{x_5},x_2),6$	$(\pi^1_{x_1},x_3),4$ $(\pi^1_{x_5},x_3),5$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
$(x_3,2)[1]$	$\pi^2_{x_2}$		$(\pi^2_{x_2},x_3),7$		
	$\pi^2_{x_3} = (\pi^1_{x_1},x_3) = (x_1,x_3)$, Coût 4				
$(x_2,3)[0]$	$\pi^2_{x_3} \to (\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_2),6$ $(\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
	$\pi^{\mathbf{3}}_{\mathbf{x_2}} = (\pi^{\mathbf{1}}_{\mathbf{x_5}},\mathbf{x_2}) = (\mathbf{x_1},\mathbf{x_4},\mathbf{x_5},\mathbf{x_2})$, Coût 6				
$(\mathbf{x_2},\mathbf{4})[\mathbf{0}]$	CheminSuivant$(x_5,2)$	$(\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
$(x_5,2)[1]$	CheminSuivant$(x_4,2)$	$(\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
$(x_4,2)[2]$	CheminSuivant$(x_1,2)$	$(\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$(\pi^1_{x_5},x_4),4$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$
$(x_1,2)[3]$	$\times$				
$(x_4,2)[2]$	$\pi^2_{x_4} = (\pi^1_{x_5},x_4),4 = (x_1,x_4,x_5,x_4)$				
$(x_5,2)[1]$	$\pi^2_{x_4} \to (\pi^2_{x_4},x_5),5$	$(\pi^2_{x_3},x_2),6$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_2},x_5),4$ $(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
	$\pi^2_{x_5} = (\pi^1_{x_2},x_5) = (x_1,x_2,x_5)$, Coût 4				
$(x_2,4)[0]$	$\pi^2_{x_5} \to (\pi^2_{x_5},x_2),7$	$(\pi^2_{x_3},x_2),6$ $(\pi^2_{x_5},x_2),7$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
	$\pi^{\mathbf{4}}_{\mathbf{x_2}} = (\pi^{\mathbf{2}}_{\mathbf{x_3}},\mathbf{x_2}) = (\mathbf{x_1},\mathbf{x_3},\mathbf{x_2})$, Coût 6				
$(\mathbf{x_2},\mathbf{5})[\mathbf{0}]$	CheminSuivant$(x_3,3)$	$(\pi^2_{x_5},x_2),7$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
$(x_3,3)[1]$	CheminSuivant$(x_1,2)$	$(\pi^2_{x_5},x_2),7$	$(\pi^1_{x_5},x_3),5$ $(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
$(x_1,2)[2]$	$\times$				
$(x_3,3)[1]$	$\pi^3_{x_3} = (\pi^1_{x_5},x_3),5 = (x_1,x_4,x_5,x_3)$				
$(x_2,5)[0]$	$\pi^3_{x_3} \to (\pi^3_{x_3},x_2),7$	$(\pi^2_{x_5},x_2),7$ $(\pi^3_{x_3},x_2),7$	$(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
	$\pi^{\mathbf{5}}_{\mathbf{x_2}} = (\pi^{\mathbf{2}}_{\mathbf{x_5}},\mathbf{x_2}) = (\mathbf{x_1},\mathbf{x_2},\mathbf{x_5},\mathbf{x_2})$, Coût 7				
$(\mathbf{x_2},\mathbf{6})[\mathbf{0}]$	CheminSuivant$(x_5,3)$	$(\pi^3_{x_3},x_2),7$	$(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^1_{x_3},x_5),5$ $(\pi^2_{x_4},x_5),5$
$(x_5,3)[1]$	$\pi^2_{x_2}$				$(\pi^2_{x_2},x_5),8$
	$\pi^3_{x_5} = (\pi^1_{x_3},x_5),5 = (x_1,x_2,x_3,x_5)$				
$(x_2,6)[0]$	$\pi^3_{x_5} \to (\pi^3_{x_5},x_2),8$	$(\pi^3_{x_3},x_2),7$ $(\pi^3_{x_5},x_2),8$	$(\pi^2_{x_2},x_3),7$	$\emptyset$	$(\pi^2_{x_4},x_5),5$ $(\pi^2_{x_2},x_5),8$
	$\pi^{\mathbf{6}}_{\mathbf{x_2}} = (\pi^{\mathbf{3}}_{\mathbf{x_3}},\mathbf{x_2}) = (\mathbf{x_1},\mathbf{x_4},\mathbf{x_5},\mathbf{x_3},\mathbf{x_2})$, Coût 7				

Table 3.4 – Application de l'algorithme REA, sans cycles sur l'origine, au graphe de la figure 3.2.

Après quoi, le troisième plus court chemin arrivant en x_2, ($\pi^1_{x_5}$) de coût 6, est trouvé. L'algorithme procède ensuite de la même façon dans les itérations suivantes. Il détermine quel est le sommet prédécesseur d'un chemin déjà calculé et appelle recursivement la fonction CheminSuivant afin de déterminer le prochain plus court chemin sur ce nœud. Ces appels récursifs peuvent s'enchaîner comme c'est le cas lors des appels de CheminSuivant($x_5, 2$), CheminSuivant($x_4, 2$), CheminSuivant($x_1, 2$) pour tous les prédécesseurs jusqu'à ce qu'un nœud pour lequel le résultat a déjà été calculé soit atteint. Alors par extensions successives de nouveaux candidats sont ajoutés et de nouveaux plus courts chemins sont déterminés. Cela peut être vu lors du calcul de CheminSuivant($x_5, 2$) et de CheminSuivant($x_2, 4$).
L'exemple s'arrête dès lors que le sixième plus court chemin à la destination soit obtenu mais il aurait bien pu continuer. Cependant, comme le graphe G_2 ne présente pas de cycles si l'on supprime les nœuds origine et destination x_1 et x_2, alors l'algorithme termine à un moment donné. Cela ne serait pas le cas sur des graphes comportant des cycles, le nombre de chemins en contenant étant alors infini.

3.2 Adaptations d'algorithmes existant au cas des réseaux de transport

Par rapport aux algorithmes de la littérature, l'adaptation aux réseaux de transport suppose la transposition de la dépendance au temps et de la contrainte sur les langages réguliers. Cette dernière n'est pas un obstable puisque la transformation du graphe et de l'automate en un graphe produit donne de facto des algorithmes intégrant cette contrainte de façon directe.

3.2.1 Adaptation de l'algorithme de Yen

Puisque l'algorithme de Yen utilise une sous-routine calculant un plus court chemin entre deux nœuds sur un graphe temporairement modifié, il est directement utilisable sur des graphes dépendant du temps et pour des cheminements contraints à un langage régulier. Pour cela, il suffit d'utiliser un algorithme DRegLC qui gère les contraintes sur les modes à la place de l'algorithme de Dijkstra usuellement utilisé comme sous-routine. S'il n'existe pas de chemin respectant les contraintes sur les modes alors l'ensemble vide est retourné et l'algorithme se poursuit par la recherche d'un autre chemin.

3.2.2 Adaptation de l'algorithme de Eppstein

L'algorithme d'Eppstein se base sur le calcul d'un arbre de plus courts chemins en parcours arrière à partir de la destination, il n'est pas possible de l'adapter aux graphes qui dépendent du temps puisque l'heure à la destination n'est pas connue. Il pourrait alors être possible d'utiliser l'algorithme K^* qui ne possède pas cette restriction. Cepen-

dant, la même problèmatique apparaît lors de la construction de l'arbre de déviations et de leur enchaînement. En effet, la classification des déviations dans la structure de tas n'est pas possible lorsque les coûts sont dynamiques puisque le décalage des coûts induit par les tables horaires introduit tout autant de déviations qu'il n'est possible de connaître qu'au moment de l'exploration.

3.2.3 Adaptation de l'algorithme REA

L'algorithme REA peut s'adapter directement au graphes de transport car l'initialisation se base sur un calcul d'arbre de plus courts chemins en parcours avant à partir de l'origine et par la suite, bien que les appels récursifs soient faits en arrière, le parcours des arcs est toujours fait vers l'avant, les heures et coût des nœuds précédent sont alors connus et les contraintes d'enchainement des modes peuvent être considérés directement via le langage régulier.

3.3 Nouvel algorithme de k plus courts chemins avec cycles

3.3.1 Principe général

De l'adaptation de l'algorithme REA nous est venu le principe de l'algorithme IEA («Iterative Enumeration Algorithm» ou algorithme d'énumération itérative). Au lieu d'énumérer les chemins récursivement depuis la destination comme c'est le cas pour REA, l'algorithme IEA énumère les chemins de façon itérative depuis l'origine. De plus, cet algorithme permet d'introduire une procèdure filtrant des cycles de longueur fixe au cours de la recherche, chose qu'il n'est pas possible d'accomplir avec l'algorithme REA.

Dans l'algorithme IEA, il faut générer et conserver plusieurs chemins. Chaque nœud comporte donc plusieurs étiquettes (ou labels) réflétant chacune un parcours antérieur différent. Afin de limiter la propagation de labels, nous utilisons la règle exploitée également dans l'algorithme REA, à savoir que pour un nœud donné, « seuls peuvent être utilisés les labels provenant des noeuds constituant le plus court chemin précédent ». Ainsi, pour un noeud donné, tant que l'étiquette de plus petit coût propagée au reste du graphe n'appartient pas à une solution, toutes les étiquettes de coût supérieur sur ce nœud ne peuvent pas fournir de solution et sont considérées comme bloquées. Globalement, l'algorithme IEA est un algorithme à fixation d'étiquettes, de type Dijkstra, dont la sélection de la prochaine étiquette à traiter ne se base pas uniquement sur le coût.
Pour mettre en œuvre ce principe, nous ajoutons des marques sur chaque nœud, dénotant le nombre de labels pouvant être couramment dépilés. Lors de l'initialisation, toutes les marques sont à 1 pour tous les noeuds, afin que l'algorithme IEA fonctionne comme l'algorithme de Dijkstra. Par la suite, à chaque fois qu'une étiquette est dépilée, la marque sur le nœud associé est décrémentée de 1. Lorsque la marque est à zéro, le sommet ne peut plus fournir de nouveaux labels candidats. C'est uniquement lorsqu'un

chemin solution passera par le nœud que le compteur augmentera à nouveau, pouvant prendre des valeurs arbitrairement grandes sur des nœuds particuliers du graphe.
Ainsi, pour déterminer plusieurs chemins, à chaque sommet x, on associe un ensemble de labels $l_x^* = \{l_x^i\}$ où l_x^i représente le label numéro i pour ce sommet x et un compteur B_x permettant de savoir combien de labels il est possible de dépiler lors des itérations de l'algorithme. Chaque label l_x^i contient un coût, l'identifiant du sommet prédécesseur ainsi que le numéro de son label. Par exemple, si $l_x^2 = (10, y, 1)$, cela traduit le fait que le sommet x a un coût 10 comme deuxième label et que ce coût a été obtenu par le sommet y en propageant son premier label.

3.3.2 Algorithme IEA

L'algorithme IEA est décrit dans le pseudo-code 5 et résout, comme il est d'usage, le sous-problème consistant à trouver le coût de chacun des k plus courts chemins sans retourner les chemins eux mêmes. L'algorithme IEA considère en entrée, un graphe G dont deux sommets distincts o et d correspondent à l'origine et la destination entre lesquelles il faut calculer k chemins. La valuation des arcs du graphe se fait par une fonction f et le coût initial sur l'origine est de c_0.
Il utilise les notations suivantes :

- $l = \bigcup_{x \in V} l_x^*$: l_x^* étant l'ensemble des étiquettes notées l_x^i pour chaque nœud x ;
- B : l'ensemble des marques notées B_x indiquant le nombre possible d'étiquettes à dépiler sur un nœud x ;
- H : l'ensemble des étiquettes candidates. Cet ensemble est implémenté par un tas ;
- S : l'ensemble des solutions, donnant pour chaque valeur de k le coût du chemin associé.

Chaque étiquette, l_x^i correspond à un triplet (c, y, j) où c est le coût du label, y l'identifiant du nœud précédent et j le numéro de label de y dont l'extension a permis d'obtenir l_x^i.
En sortie, l'algorithme IEA fournit l'ensemble des solutions S composé de paires (i, c) où i est le numéro du chemin et c son coût.

Les lignes 2 à 5 correspondent à l'étape d'initialisation : les ensembles de labels sont vides et les compteurs de marque sont initialisés à 1 pour tous les noeuds, l'étiquette initiale de coût c_0 est créée et insérée dans le tas, enfin, l'ensemble des solutions S est vide.
Ensuite tant que le tas n'est pas vide, (dans le cas d'un épuisement de l'énumération qui peut se produire sur des graphes triviaux ne contenant pas de cycles) et que le nombre de chemins demandés n'est pas atteint, il faut récupérer le label de meilleur coût dont la marque est au moins égale à 1 (ligne 7), supprimer ce label de la liste des labels candidats et décrémenter sa marque.

Algorithme 5 Iterative Enumeration Algorithm

Requis $G = (V, E)$, $f : E \to \mathbb{R}_+$

1: **function** IEA(G, $o \in V$, $d \in V$, $k \in \mathbb{N}$, f, $c_0 \in \mathbb{R}_+$, $n \in \mathbb{N}$) :$S \in \mathbb{N} \times \mathbb{R}$
2: $B_x \leftarrow 1 \; \forall x \in V$
3: $l_o^1 \leftarrow (c_0, \emptyset, 0)$
4: $H \leftarrow \{l_o^1\}$
5: $S \leftarrow \emptyset$
6: **Tant que** $H \neq \emptyset \wedge |S| \neq k$ **faire**
7: $l_x^i = (c, w, k) \leftarrow min_{B_z \geq 1}(H = \{l_z^*\})$
8: $H \leftarrow H \setminus \{l_x^i\}$
9: $B_x \leftarrow B_x - 1$
10: **Si** $x \neq d$ **alors**
11: **Pour** $(x, y) \in \{(x, z) \in E \mid z \neq o\}$ **faire**
12: $j \leftarrow |l_y^*| + 1$
13: $l_y^j \leftarrow (c + f((x, y)), x, i)$
14: $H \leftarrow H \cup \{l_y^j\}$
15: **Fin Pour**
16: **Sinon**
17: $S \leftarrow S \cup \{(|S| + 1, c)\}$
18: **Tant que** $w \neq o$ **faire**
19: $B_w \leftarrow B_w + 1$
20: $(-, w, k) \leftarrow l_w^k$
21: **Fin Tant que**
22: $B_o \leftarrow B_o + 1$
23: **Fin Si**
24: **Fin Tant que**
25: **return** S
26: **Fin function**

Si la destination n'est pas atteinte, alors chaque successeur **différent** de l'origine, **pour éviter les cycles à l'origine**, est inséré dans l'ensemble des labels candidats H (lignes 10 à 14).
Dans le cas où le nœud traité est la destination, c'est qu'une nouvelle solution a été touvée. La marque B_y de chacun des nœuds y présents dans le chemin obtenu est alors incrémentée (lignes 18 à 22), libérant une étiquette supplémentaire sur chacun d'eux pour la prochaine étape.

L'algorithme IEA permet ainsi de calculer les k plus courts chemins d'un sommet origine vers un sommet destination. Les chemins retournés peuvent comporter des cycles mais pas sur l'origine. Il est à noter que, comme pour l'algorithme REA, l'algorithme IEA permet aussi de trouver $k_x \preceq k$ chemins pour chaque nœud x du graphe. Le fonctionnement de cet algorithme est illustré dans l'exemple détaillé ci-après.

3.3.3 Exemple illustratif de l'application de l'algorithme IEA

Prenons comme exemple le graphe G_3 rappelé dans la figure 3.3 sur lequel nous allons appliquer l'algorithme IEA pour calculer les plus courts chemins élémentaires de x_1 à x_2.

Sur cet exemple, les chemins élémentaires que l'on peut déterminer, par ordre crois-

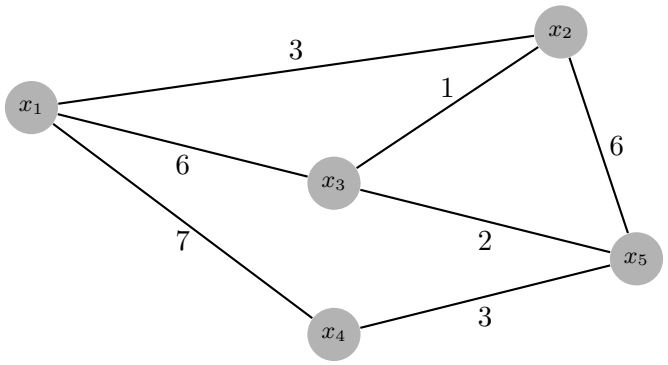


FIGURE 3.3 – Graphe exemple G_3

sant de coût, sont au nombre de cinq et sont les suivants :

- $\pi^1 = (x_1, x_2)$, $c(\pi^1) = 3$
- $\pi^2 = (x_1, x_3, x_2)$, $c(\pi^2) = 7$
- $\pi^3 = (x_1, x_4, x_5, x_3, x_2)$, $c(\pi^3) = 13$
- $\pi^4 = (x_1, x_3, x_5, x_2)$, $c(\pi^4) = 14$
- $\pi^5 = (x_1, x_4, x_5, x_2)$, $c(\pi^5) = 16$

Lors de l'étape d'initialisation une étiquette pour le sommet origine x_1, $l^1_{x_1} = (c_0, \emptyset, 0)$ de coût c_0 est empilée dans le tas H représentant l'ensemble des étiquettes candidates. Les autres sommets n'ont pas de labels. Le compteur B_x associé à chaque nœud est initialisé à 1. Dans le tableau 3.5 sont présentés, pour chaque tour de boucle de l'algorithme, les compteurs ainsi que les labels non encore utilisés de chacun des nœuds du graphe.
Lors de la première étape de l'algorithme (première exécution de la boucle), l'étiquette de plus petit coût, ie. $l^1_{x_1}$ est dépilée et les nœuds successeurs sont évalués. La première ligne du tableau 3.5 (Etape 1) donne donc l'état des labels et des compteurs de chaque nœud après exécution d'une itération de la boucle principale. L'extension du nœud x_1 a ainsi permet de créer des étiquettes pour les nœuds x_2, x_3 et x_4 avec des coûts respectifs 3, 6 et 7, avec la même valeur de prédécesseur $(x_1, 1)$. On peut remarquer que le nœud x_1, correspondant à l'origine, ne sera plus utilisé par la suite dans l'algorithme car les boucles sur l'origine sont éliminées, il n'est donc pas présent dans le tableau.

Lors de la deuxième étape, c'est l'étiquette de coût 3 associée au sommet de destination x_2 qui est sélectionnée. Elle donne le chemin π'^1 passant par x_1 et x_2 correspondant au premier chemin sans cycle π^1. Le compteur B_{x_2} qui venait d'être décrémenté est alors incrémenté comme affiché dans la dernière ligne (Etape 2).
Lors de la troisième étape, l'étiquette de coût 6 est sélectionnée (associée au sommet x_3), le compteur B_{x_3} est mis à 0 puis l'étiquette est propagée vers les successeurs de x_3, c'est à dire x_2 et x_5. Cela produit un second label pour le sommet x_2 et un premier label pour le sommet x_5 en provenance du label $(x_3, 1)$.
Au cours de la quatrième étape, l'étiquette de coût 7 du sommet x_4 est sélectionnée

Etape	x_2		x_3		x_4		x_5	
	B_{x_2}	$l^*_{x_2}$	B_{x_3}	$l^*_{x_3}$	B_{x_4}	$l^*_{x_4}$	B_{x_5}	$l^*_{x_5}$
1	1	$l^1_{x_2} : (3, x_1, 1)$	1	$l^1_{x_3} : (6, x_1, 1)$	1	$l^1_{x_4} : (7, x_1, 1)$	1	
2	1	~~$(3, x_1, 1)$~~	1	$(6, x_1, 1)$	1	$(7, x_1, 1)$	1	
	0		1	$(6, x_1, 1)$	1	$(7, x_1, 1)$	1	
	$\pi'^1 : (x_1, x_2) : \pi^1$							
	1		1	$(6, x_1, 1)$	1	$(7, x_1, 1)$	1	
3	1		1	~~$(6, x_1, 1)$~~	1	$(7, x_1, 1)$	1	
	1		**0**		1	$(7, x_1, 1)$	1	
	1	$l^2_{x_2} : (7, x_3, 1)$	0		1	$(7, x_1, 1)$	1	$l^1_{x_5} : (8, x_3, 1)$
4	1	$(7, x_3, 1)$	0		1	~~$(7, x_1, 1)$~~	1	$(8, x_3, 1)$
	1	$(7, x_3, 1)$	0		**0**		1	$(8, x_3, 1)$
	1	$(7, x_3, 1)$	0		0		1	$(8, x_3, 1)$ $l^2_{x_5} : (10, x_4, 1)$
5	1	~~$(7, x_3, 1)$~~	0		0		1	$(8, x_3, 1)$ $(10, x_4, 1)$
	0		0		0		1	$(8, x_3, 1)$ $(10, x_4, 1)$
	$\pi'^2 : (x_1, x_3, x_2) : \pi^2$							
	1		**1**		0		1	$(8, x_3, 1)$ $(10, x_4, 1)$

TABLE 3.5 – Evolution des étiquettes en début de recherche lors de l'application de IEA sur le graphe G_1

tout comme aurait pu l'être de façon totalement indifférente celle de coût identique du sommet x_2. Puis le compteur B_{x_4} est mis à 0 puis l'étiquette est propagée vers le sommet x_5, successeur de x_4 et produit un second label pour ce sommet.
A la cinquième étape, l'étiquette de coût 7 du sommet x_2 est sélectionnée et fournit alors un second chemin π'^2 passant par x_1, x_3 puis x_2 et correspondant au deuxième chemin sans cycle π^2. Les compteurs de ces deux derniers sommets (x_2 et x_3) sont alors incrémentés de 1 (rappelons que x_1 est ignoré).

A partir des étapes suivantes, pour simplifier, les tableaux affichent chaque étape sur une unique ligne, l'étiquette dépilée et barrée, celles en gras sont les nouveaux labels insérés, le chiffre en gras est la marque décrémentée ; comme illustré sur le tableau 3.6.

Etape	x_2		x_3		x_4		x_5	
	B_{x_2}	$l^*_{x_2}$	B_{x_3}	$l^*_{x_3}$	B_{x_4}	$l^*_{x_4}$	B_{x_5}	$l^*_{x_5}$
5	1		1		0		1	$(8, x_3, 1)$ $(10, x_4, 1)$
6	1	$\mathbf{l^3_{x_2} : (14, x_5, 1)}$	1	$\mathbf{l^2_{x_3} : (10, x_5, 1)}$	0	$\mathbf{l^2_{x_4} : (11, x_5, 1)}$	**0**	~~$(8, x_3, 1)$~~ $(10, x_4, 1)$
7	1	$(14, x_5, 1)$ $\mathbf{l^4_{x_2} : (11, x_3, 2)}$	**0**	~~$(10, x_5, 1)$~~	0	$(11, x_5, 1)$	0	$(10, x_4, 1)$ $\mathbf{l^3_{x_5} : (12, x_3, 2)}$

TABLE 3.6 – Suite de l'évolution des étiquettes lors de l'application de IEA sur le graphe G_1

Lors du passage du tableau 3.6 au tableau 3.7, s'applique pour la première fois la

condition sur les compteurs puisque c'est le label de coût 11 du nœud x_2 qui est traité dans l'étape 8 et pas celui de coût 10 du nœud x_5 car le compteur de x_5 vaut 0. Lors de cette étape, un nouveau chemin π'^3 est obtenu. Il passe deux fois par le nœud x_3 dont le compteur passe alors à la valeur 2 alors que les compteurs des autres nœuds du chemin passent à 1.

Etape	x_2		x_3		x_4		x_5	
	B_{x_2}	$l^*_{x_2}$	B_{x_3}	$l^*_{x_3}$	B_{x_4}	$l^*_{x_4}$	B_{x_5}	$l^*_{x_5}$
7	1	$(14, x_5, 1)$ $(11, x_3, 2)$	0		0	$(11, x_5, 1)$	0	$(10, x_4, 1)$ $(12, x_3, 2)$
8	**0**	$(14, x_5, 1)$ ~~$(11, x_3, 2)$~~	0		0	$(11, x_5, 1)$	0	$(10, x_4, 1)$ $(12, x_3, 2)$
	$\pi'^3 : (x_1, x_3, x_5, x_3, x_2)$							
9	1	$(14, x_5, 1)$ $\mathbf{l^5_{x_2}} : (\mathbf{16}, \mathbf{x_5}, \mathbf{2})$	2	$\mathbf{l^3_{x_3}} : (\mathbf{12}, \mathbf{x_5}, \mathbf{2})$	0	$(11, x_5, 1)$ $\mathbf{l^3_{x_4}} : (\mathbf{13}, \mathbf{x_5}, \mathbf{2})$	~~1~~ **0**	~~$(10, x_4, 1)$~~ $(12, x_3, 2)$
10	1	$(14, x_5, 1)$ $(16, x_5, 2)$ $\mathbf{l^6_{x_2}} : (\mathbf{13}, \mathbf{x_3}, \mathbf{3})$	**1**	~~$(12, x_5, 2)$~~	0	$(11, x_5, 1)$ $(13, x_5, 2)$	0	$(12, x_3, 2)$ $\mathbf{l^4_{x_5}} : (\mathbf{14}, \mathbf{x_3}, \mathbf{3})$
11	**0**	$(14, x_5, 1)$ $(16, x_5, 2)$ ~~$(13, x_3, 3)$~~	1		0	$(11, x_5, 1)$ $(13, x_5, 2)$	0	$(12, x_3, 2)$ $(14, x_3, 3)$
	$\pi'^4 : (x_1, x_4, x_5, x_3, x_2) : \pi^3$							
12	1	$(14, x_5, 1)$ $(16, x_5, 2)$	2		~~1~~ **0**	~~$(11, x_5, 1)$~~ $(13, x_5, 2)$	1	$(12, x_3, 2)$ $(14, x_3, 3)$ $\mathbf{l^5_{x_5}} : (\mathbf{14}, \mathbf{x_4}, \mathbf{2})$
13	1	$(14, x_5, 1)$ $(16, x_5, 2)$ $\mathbf{l^7_{x_2}} : (\mathbf{18}, \mathbf{x_5}, \mathbf{3})$	2	$\mathbf{l^4_{x_3}} : (\mathbf{14}, \mathbf{x_5}, \mathbf{3})$	0	$(13, x_5, 2)$ $\mathbf{l^4_{x_4}} : (\mathbf{15}, \mathbf{x_5}, \mathbf{3})$	**0**	~~$(12, x_3, 2)$~~ $(14, x_3, 3)$ $(14, x_4, 2)$
14	**0**	~~$(14, x_5, 1)$~~ $(16, x_5, 2)$ $(18, x_5, 3)$	2	$(14, x_5, 3)$	0	$(13, x_5, 2)$ $(15, x_5, 3)$	0	$(14, x_3, 3)$ $(14, x_4, 2)$
	$\pi'^5 : (x_1, x_3, x_5, x_2) : \pi^4$							
15	1	$(16, x_5, 2)$ $(18, x_5, 3)$	3	$(14, x_5, 3)$	0	$(13, x_5, 2)$ $(15, x_5, 3)$	1	$(14, x_3, 3)$ $(14, x_4, 2)$

TABLE 3.7 – Suite de l'évolution des étiquettes lors de l'application de IEA sur le graphe G_1

Lors des étapes 9, 10 et 11, les labels de coûts 10 (nœud x_5), 12 (nœud x_3) et 13 (nœud x_2) sont sélectionnés. Un quatrième chemin est obtenu π'^4 correspondant au troisième chemin élémentaire π^3 et les compteurs des nœuds composant ce chemin sont incrémentés. Les étapes de 12 à 14 permettent ensuite d'obtenir un nouveau chemin de coût 14 qui correspond au chemin élémentaire π^4. La dernière ligne présente l'actualisation des compteurs de marque sur chacun des nœuds.
Le tableau 3.8 donne les dernières itérations de l'algorithme IEA : les étapes 15 à 17 permettent d'obtenir un chemin avec cycle, puis les étapes de 18 à 19 calculent le dernier chemin élémentaire π^5.

On peut noter que, sans condition d'arrêt, l'algorithme IEA poursuit la recherche de chemin avec cycles et peut en fournir un nombre arbitrairement grand puisque le nombre de chemins avec cycles est infini dès lors qu'il en existe au moins un.

Etape	x_2		x_3		x_4		x_5	
	B_{x_2}	$l^*_{x_2}$	B_{x_3}	$l^*_{x_3}$	B_{x_4}	$l^*_{x_4}$	B_{x_5}	$l^*_{x_5}$
15	1	$(16, x_5, 2)$ $(18, x_5, 3)$ $\mathbf{l^8_{x_2} : (20, x_5, 4)}$	3	$(14, x_5, 3)$ $\mathbf{l^5_{x_3} : (16, x_5, 4)}$	0	$(13, x_5, 2)$ $(15, x_5, 3)$ $\mathbf{l^5_{x_4} : (17, x_5, 4)}$	**0**	~~$(14, x_3, 3)$~~ $(14, x_4, 2)$
16	1	$(16, x_5, 2)$ $(18, x_5, 3)$ $(20, x_5, 4)$ $\mathbf{l^9_{x_2} : (15, x_3, 4)}$	**2**	~~$(14, x_5, 3)$~~ $(16, x_5, 4)$	0	$(13, x_5, 2)$ $(15, x_5, 3)$ $(17, x_5, 4)$	0	$(14, x_4, 2)$ $\mathbf{l^6_{x_5} : (16, x_3, 4)}$
17	**0**	$(16, x_5, 2)$ $(18, x_5, 3)$ $(20, x_5, 4)$ ~~$(15, x_3, 4)$~~	2	$(16, x_5, 4)$	0	$(13, x_5, 2)$ $(15, x_5, 3)$ $(17, x_5, 4)$	0	$(14, x_4, 2)$ $(16, x_3, 4)$
	$\pi'^6 : (x_1, x_3, x_5, x_3, x_5, x_3, x_2)$							
18	1	$(16, x_5, 2)$ $(18, x_5, 3)$ $(20, x_5, 4)$ $\mathbf{l^{10}_{x_2} : (20, x_5, 5)}$	5	$(16, x_5, 4)$ $\mathbf{l^6_{x_3} : (16, x_5, 5)}$	0	$(13, x_5, 2)$ $(15, x_5, 3)$ $(17, x_5, 4)$ $\mathbf{l^6_{x_4} : (17, x_5, 5)}$	~~2~~ **1**	~~$(14, x_4, 2)$~~ $(16, x_3, 4)$
19	**0**	~~$(16, x_5, 2)$~~ $(18, x_5, 3)$ $(20, x_5, 4)$ $(20, x_5, 5)$	5	$(16, x_5, 4)$ $(16, x_5, 5)$	0	$(13, x_5, 2)$ $(15, x_5, 3)$ $(17, x_5, 4)$ $(17, x_5, 5)$	1	$(16, x_3, 4)$
	$\pi'^7 : (x_1, x_4, x_5, x_2) : \pi^5$							
...	1	$(18, x_5, 3)$ $(20, x_5, 4)$ $(20, x_5, 5)$	5	$(16, x_5, 4)$ $(16, x_5, 5)$	1	$(13, x_5, 2)$ $(15, x_5, 3)$ $(17, x_5, 4)$ $(17, x_5, 5)$	2	$(16, x_3, 4)$

TABLE 3.8 – Fin de l'évolution des étiquettes lors de l'application de IEA sur le graphe G_1

L'algorithme IEA (de même que l'algorithme REA) génère de nombreux chemins avec cycles. Dans l'application visée, de tels chemins ne sont pas pertinents et sont considérés comme non valides. Or dans les algorithmes IEA et REA, les chemins avec cycles sont seulement détectés a posteriori. Nous allons présenter dans la section suivante comment éliminer certains cycles avant d'arriver au sommet de destination afin d'accélérer la recherche de chemins élémentaires.

3.4 Introduction de procédures coupe-cycle

Afin de limiter l'obtention de chemins avec cycles nous avons tenté de supprimer le plus de cycles possible au moment de la recherche, et en particulier les cycles générés au début de la recherche, proches de l'origine. En effet, un cycle se produisant tôt a plus de possibilités de produire des chemins jusqu'à la destination. Ainsi, plus les cycles sont détectés tôt, plus on évite de propager inutilement des labels n'ayant aucune chance de

produire un chemin élémentaire.

3.4.1 Coupe cycle pour l'algorithme REA

L'algorithme REA a été adapté afin de couper les cycles qui se forment à l'origine lors de l'appel à la méthode CheminSuivant en ce nœud avec une valeur de k plus grande que 1. Cependant, il n'est pas possible d'étendre cette adaptation à n'importe quel nœud du graphe comme nous allons l'exposer par la suite.
Admettons que lors de l'appel à CheminSuivant(x, k) sur un nœud x quelconque du graphe hormis l'origine, le chemin extrait $(\pi_y^{k'}, x)$ tel que $\pi_x^{k-1} = (\pi_y^{k'-1}, x)$ contienne un cycle. Si l'appel termine sans résultat alors $\pi_y^{k'}$ n'est jamais déterminé. Cela est légitime si tous les chemins après CheminSuivant$(y, k' - 1)$ contiennent des cycles mais il est impossible de le déterminer car un chemin avec cycle va masquer tous les chemins sans cycles qui le suivent. Il faut donc nécessairement trouver ce chemin mais il est alors possible que tous les prédécesseurs de y soient dans la même situation, ne serait ce que si x se trouve parmis les prédécesseurs de y à une distance telle que les cycles ne soient pas détectés aussi loin. Comme il est irréalisable en pratique de vérifier à chaque fois que la totalité du chemin ne contient pas de cycles, cette situation peut se produire. L'algorithme est alors bloqué et aucun chemin ne peut être trouvé alors qu'il reste des chemins élémentaires à découvrir.
Il n'est donc pas possible de supprimer des cycles de longueur finie dans l'algorithme REA sans risquer un terminaison erronée à cause de son évaluation paresseuse des chemins venant des prédécesseurs d'un nœud.

3.4.2 Coupe cycle pour l'algorithme IEA

Contrairement à l'algorithme REA, pour l'algorithme IEA, couper des cycles est assez direct. Avant d'étendre un label sur un sommet, il suffit de vérifier que ce nœud n'a pas déjà été visité par le sous-chemin en cours. Cette vérification peut être limitée à un nombre fixe de sommets le long du sous-chemin, afin de réguler le temps passé à couper les cycles. En effet, dans le pire des cas, la complexité de la procédure cherchant les cycles est linéaire en fonction du nombre de nœuds du graphe. On parlera alors de procédure coupe cycles de longueur 1 lorsqu'elle se limite au premier prédecesseur.
L'algorithme 6 permet de détecter les cycles de longueur n au maximum à partir d'un sommet y et du label de son prédécesseur l_x^i (et connaissant le nœud origine o).

Pour utiliser cette procédure de coupe-cycle dans l'algorithme IEA, il convient d'insérer la condition

$$x = o \vee \neg contientcycle(l, x, i, y, n, o)$$

à la ligne 13 de l'algorithme 5 pour n'exécuter les lignes 13 et 14 que si le nœud à étendre est l'origine (pour la première itération) ou si le nœud successeur n'engendre pas de cycles de longueur inférieure à n. L'algorithme ainsi obtenu sera noté IEA-C$_n$.

Algorithme 6 Procédure coupe cycles

Requis : $G = (V, E)$

function *contientcycle*$(l, x \in V, i \in \mathbb{N}, y \in V, n \in \mathbb{N}, o \in V)$:booléen
  Tant que $n > 0$ **faire**
    $(-, x, i) \leftarrow l_x^i$
    Si $x = y$ **alors**
      return Vrai
    Fin Si
    $n \leftarrow n - 1$
  Fin Tant que
  return Faux
Fin function

Cependant, une telle procédure a un effet sur la correction de l'algorithme. Il se peut désormais que les chemins ne soient plus énumérés dans l'ordre croissant de coût comme montré à la section 3.4.2.1.

3.4.2.1 Exemple illustratif du non respect des coûts croissants pour IEA-C$_1$

Reprenons l'exécution de IEA à l'étape 5 (en fin du tableau 3.5), puis utilisons IEA-C$_1$ à la place en interdisant les cycles de longueur 1, le tableau 3.9 récapitule l'évolution des labels. Dans cet exemple, à l'étape 7, le chemin π^4 de coût 14 est obtenu alors que le chemin π^3 de coût 13 n'est obtenu qu'à l'étape 10. En effet, lors de l'étape 6, l'étiquette de coût 8 du nœud x_5 produit un cycle de longeur 1 sur x_3. Ce nœud ne reçoit donc pas de nouveau label et ne peut donc pas être étendu vers le nœud de destination ce qui aurait permis d'augmenter le compteur sur le nœud x_5 et de permettre l'extension du label de coût 10 et d'aboutir ainsi au chemin π^3.

Etape	x_2		x_3		x_4		x_5	
	B_{x_2}	$l^*_{x_2}$	B_{x_3}	$l^*_{x_3}$	B_{x_4}	$l^*_{x_4}$	B_{x_5}	$l^*_{x_5}$
5	1		1		0		1	$(8, x_3, 1)$ $(10, x_4, 1)$
6	1	$\mathbf{l^3_{x_2}} : (\mathbf{14}, \mathbf{x_5}, \mathbf{1})$	1		0	$\mathbf{l^2_{x_4}} : (\mathbf{11}, \mathbf{x_5}, \mathbf{1})$	**0**	~~$(8, x_3, 1)$~~ $(10, x_4, 1)$
7	**0**	~~$(14, x_5, 1)$~~	1		0	$(11, x_5, 1)$	0	$(10, x_4, 1)$
	$\pi'^3 : (x_1, x_3, x_5, x_2) : \pi^4$							
8	1	$\mathbf{l^4_{x_2}} : (\mathbf{16}, \mathbf{x_5}, \mathbf{2})$	2	$\mathbf{l^2_{x_3}} : (\mathbf{12}, \mathbf{x_5}, \mathbf{2})$	0	$(11, x_5, 1)$	~~1~~ **0**	~~$(10, x_4, 1)$~~
9	1	$(16, x_5, 2)$ $\mathbf{l^5_{x_2}} : (\mathbf{13}, \mathbf{x_3}, \mathbf{2})$	**1**	~~$(12, x_5, 2)$~~	0	$(11, x_5, 1)$	0	
10	**1**	$(16, x_5, 2)$ ~~$(13, x_3, 2)$~~	1		0	$(11, x_5, 1)$	0	
	$\pi'^4 : (x_1, x_4, x_5, x_3, x_2) : \pi^3$							
...	2	$(16, x_5, 2)$	2		1	$(11, x_5, 1)$	1	

TABLE 3.9 – Evolution des étiquettes lors de la recherche avec coupe cycles

Ainsi, l'introduction de la procédure de coupe-cycle modifie donc l'ordre d'appari-

tion des chemins qui ne sont plus déterminés par coût croissant. Or, lors de l'appel de l'algorithme IEA, avec ou sans coupe-cycle, il est nécessaire de fixer un nombre maximum de chemins avec cycle à générer (sinon l'algorithme boucle à l'infini). L'ajout de la procédure coupe-cycle fait donc perdre la garantie d'obtenir l'ensemble des plus courts chemins car un chemin de coût inférieur au dernier retourné peut être en attente.

3.5 Implémentations et analyses expérimentales

3.5.1 Contexte expérimental

Les expérimentations se basent sur le réseau de transport de Toulouse compilé en graphe via la plateforme *PlayMob'* (voir chapitre 5). La voirie *OpenStreetMap* a été obtenue avec *GeoFabrik*[4] tandis que le réseau de transport, au format *GTFS*, émane de la métropole de Toulouse. Le graphe final contient 75 837 nœuds et 484 426 arcs dont 43 318 arcs de transport en commun. Les modes de transport considérés sont l'ensemble des transports en commun ainsi que la marche. Les contraintes d'utilisation des modes de transport sont représentés par un automate acceptant l'ensemble des chemins (pas de restriction sur les modes de transport) et l'heure de départ est fixée à 9h du matin. Un ensemble de 40 instances correspondant à des paires (origine, destination) a été généré aléatoirement.
Les expérimentations menées visent à comparer les performances des algorithmes Yen, REA, IEA, IEA-C$_n$ pour obtenir un ensemble de k plus courts chemins élémentaires. Pour cela, nous avons fait varier le nombre de chemins élémentaires de 10 à 100 pour l'algorithme de Yen, produisant directement des chemins élémentaires. Pour les algorithmes IEA et IEA-C$_n$ et REA, une procédure de filtrage des chemins avec cycles est utilisée pour compter le nombre de chemins élémentaires obtenus parmi l'ensemble des chemins avec cycles. Pour ces algorithmes, nous avons fait varier le nombre de chemins avec cycles de 100 à 600 avec des coupes cycles de longueur 1 à 3.
Tous les algorithmes (Yen, REA, IEA, IEA-C$_n$) ont été implémentés dans la bibliothèque *C++geroli* compilé par GCC. Les expérimentations ont été faites sur une machine sous Linux 3.2.0.4-amd64 avec un processeur Intel(R) Core(TM) i5-3337CPU @ 1.7GHz ayant respectivement 1664Ko et 3Go de mémoires cache et principale.

3.5.2 Résultats et analyses

3.5.2.1 Calcul des k plus courts chemins sans cycle

Pour l'algorithme de Yen, les résultats sont regroupés dans le tableau 3.10. Y sont affichés pour chaque valeur de k, le temps de calcul en millisecondes (temps), le coût moyen (coût) en secondes et le nombre moyen de sommets composant le chemin (sauts) ainsi que les écarts-types $\sigma(coût)$ et $\sigma(sauts)$. De plus, puisque l'algorithme de Yen appelle comme sous-routine l'algorithme $DReg_{LC}$ exactement le nombre de fois qu'il y a de

nœuds dans le chemin précédent, le temps de calcul est limité par le nombre moyen de nœuds dans les chemins multiplié par le temps moyen nécessaire pour résoudre $DReg_{LC}$ (en moyenne 56ms pour les instances considérées). Cette information apparaît dans la dernière colonne du tableau. On observe que le coût moyen des chemins augmente légèrement en fonction du nombre de chemins demandés ce qui est tout à fait normal puisque le coût des chemins augmente avec l'augmentation de k. L'écart type du coût augmente également signifiant que le coût des chemins solution devient de plus en plus diversifié lorsque le nombre de chemin augmente. En ce qui concerne le temps d'exécution, on retrouve la complexité théorique linéaire en k puisque le temps de calcul passe de 24 secondes pour 10 chemins à 248 secondes pour 100.

k	temps(ms)	coût(s)	$\sigma(c)$	sauts	$\sigma(sauts)$	temps max(ms)
10	23 849	81.403	0.015	90.65	4.16	50 712
20	49 276	81.404	0.093	91.17	4.44	102 012
30	76 052	81.406	0.155	91.63	4.51	153 787
40	100 371	81.407	0.171	91.84	4.36	205 532
50	125 666	81.408	0.199	91.93	4.47	257 166
100	248 875	81.412	0.301	92.88	4.53	519 595

TABLE 3.10 – Résultats pour l'extension de l'algorithme de Yen

3.5.2.2 Calcul des k plus courts chemins avec cycles

En ce qui concerne les algorithmes calculant des chemins avec cycles, les résultats sont présentés dans le tableau 3.11. Comme il n'est pas possible de garantir l'obtention de tous les chemins sans cycles voulus, les algorithmes cherchent k' chemins, qui sont ensuite analysés afin de savoir combien parmi eux ne comportent pas de cycles. Cela permet de fournir le nombre k de chemins élementaires obtenus et qui peut être inférieur au nombre voulu. Les versions REA-0 et IEA-0 correspondent aux deux algorithmes sans coupe cycle. La version REA-1 correspond à une version modifiée de REA qui ne boucle pas sur l'origine. Cependant les temps d'exécution se sont avérés similaires. De ces résultats nous pouvons déduire que ces algorithmes ne peuvent pas être utilisés pour calculer des chemins sans cycles puisque le nombre de chemins élémentaires obtenus est très faible et, bien que les temps de calculs soient très bons, ils ne compensent pas l'explosion combinatoire attendue du nombre de chemins avec cycles entre deux chemins élémentaires.

En revanche, lorsqu'une procédure coupe cycles est utilisée dans l'algorithme IEA, le nombre de chemins élémentaires devient acceptable. Les temps de calcul sont meilleurs que ceux de l'algorithme de Yen et, bien que l'algorithme devienne heuristique, l'écart des moyennes des coûts des chemins est limité.

Nb	REA-0/REA-1				IEA-0			
k'	k	temps(ms)	coût(min)	$\sigma(co\hat{u}t)$	k	temps(ms)	coût(min)	$\sigma(co\hat{u}t)$
100	1.18	205.8	81.40	0.0	1.13	589	81.40	0.0
200	1.18	260.5	81.40	0.0	1.15	815.5	81.40	0.0
300	1.18	251.5	81.40	0.0	1.15	1174.3	81.40	0.0
400	1.18	256.8	81.40	0.0	1.15	1818.5	81.40	0.0

TABLE 3.11 – Résultats pour les algorithmes produisant des cycles

Nb	IEA-1			
k'	k	temps(ms)	coût(min)	$\sigma(co\hat{u}t)$
100	69.78	511	81.409	0.33
200	124.1	667	81.415	0.52
300	172.4	854	81.417	0.56
400	222.4	998	81.424	1.38
500	271.7	1330	81.434	1.79
600	319.0	1659	81.439	1.89
Nb	IEA-2			
k'	k	temps(ms)	coût(min)	$\sigma(co\hat{u}t)$
100	80	512	81.412	0.39
200	143	622	81.416	0.54
300	205	786	81.421	0.78
400	263	949	81.434	1.71
500	322	1191	81.441	1.87
600	377	1698	81.447	1.94
Nb	IEA-3			
k'	k	temps(ms)	coût(min)	$\sigma(co\hat{u}t)$
100	91	581	81.415	0.50
200	174	739	81.424	0.74
300	252	892	81.436	1.81
400	327	1015	81.454	2.97
500	401	1171	81.469	3.45
600	469.8	1577	81.475	3.51

TABLE 3.12 – Impacts de la procédure de coupe cycles

3.6 Chemins alternatifs

L'approche proposée dans ce manuscrit pour la génération d'alternatives au plus court chemin se base sur le calcul de chemins élémentaires allant de l'origine à la destination par coût croissant. Une fois ces chemins obtenus, un mécanisme de filtrage peut être mis en place pour sélectionner un ensemble de chemins pertinents pour un usager, l'ensemble obtenu représente alors différentes alternatives de trajet. Les mécanismes de filtrage doivent permettre de déterminer un ensemble de chemins suffisamment différents entre eux afin d'offrir une ou plusieurs alternatives pertinentes pour un usager.
L'approche retenue se base sur l'hypothèse qu'il n'est pas possible pour les personnes écrivant les algorithmes de savoir a priori comment caractériser les différences entre deux chemins pour n'importe quel usager potentiel, a fortiori dans un réseau multimodal. En effet, bien qu'il soit possible d'effectuer des suppositions quand à la structure de chemins alternatifs, comme les arcs ou nœuds empruntés, il ne paraît pas envisageable de fixer les conditions, qui réunies ensemble, forment un critère de décision universel sur la similarité ou la différence de chemins élémentaires. Cette analyse des chemins est alors laissée à la seconde phase de l'approche, de façon générique, pour que des choix personnalisés a posteriori soient possibles.

3.6.1 Principe

Quelques suppositions sont faites sur le cadre qui permet de sélectionner des chemins alternatifs. En premier lieu, le plus court chemin est toujours considéré comme étant la première solution à fournir car il s'agit d'un chemin remarquable. Ensuite, les chemins élémentaires sont énumérés les uns après les autres, par coût croissant, jusqu'à en trouver un qui satisfait aux exigences de l'utilisateur et qui est alors placé dans l'ensemble des solutions. Pour cela, une fonction mesurant la distance entre deux chemins est utilisée. Un chemin candidat est ajouté à l'ensemble des chemins solutions si sa distance avec chacun des chemins déjà présents dans l'ensemble solution est supérieur ou égal à un seuil fixé.
Les travaux de Chen [28] se basent également sur un ensemble de chemins originels mais qui sont tous précalculés alors que dans notre approche de nouveaux chemins peuvent arriver si jamais les chemins précédents ne sont pas retenus comme des alternatives viables. De plus, la seconde phase cherche à optimiser la sélection des alternatives là où nous cherchons à les énumérer de façon progressive par coût croissant et à prendre le premier d'entre eux qui est valide.

3.6.2 Exemple

Le graphe G_4 de la figure 3.4 est donné à titre d'exemple. Y sont considérés de manière simplifiée des lignes de bus (b_1, b_2), de métro (s_1, s_2) et de la marche (w). Les coûts associés au parcours d'un arc sont statiques et donnés après le mode le long de

l'arc. De plus, la couleur des arcs dénote leur position géographique respective, le rouge pour le Nord, le vert pour le centre et le bleu pour le Sud.

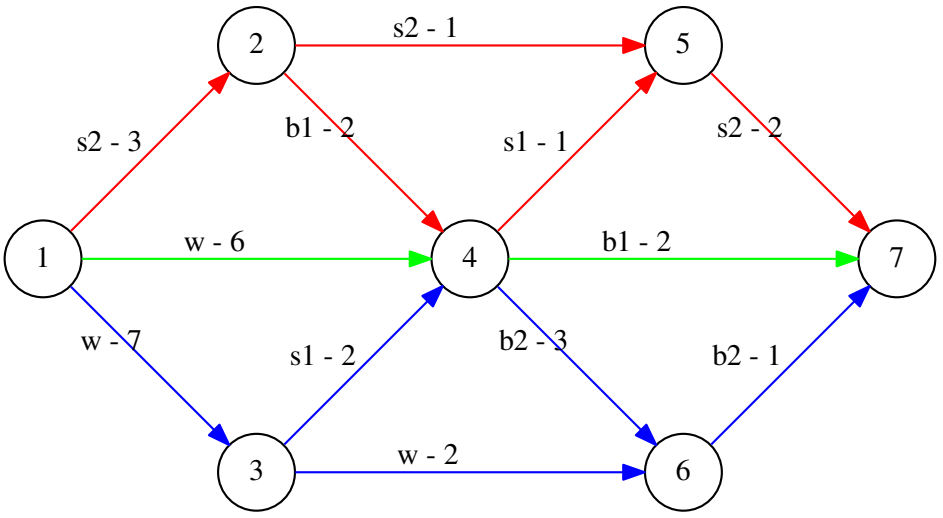


FIGURE 3.4 – Graphe exemple G_4

Trois types de profils sont établis pour les préférences de parcours et mettent l'accent sur les choix suivants d'utilisateurs :

- A considère les lignes utilisées dans chacun des modes ;
- B considère le type de transport utilisé ;
- C considère les zones géographiques empruntées.

Le tableau 3.13 présente la caractérisation des 9 chemins sans cycles existant dans le graphe G_3 de la figure 3.13 pour ces trois profils d'usager pour un trajet du sommet 1 au sommet 7. Dans ce tableau, la colonne «mode» fournit la séquence des modes de chaque chemin avec le numéro de ligne utilisée tandis que la colonne «localisation» fournit la position géographique de chacun des arcs du chemin. Enfin les colonnes «profils» présentent la représentation du chemin pour les différents usagers en fonction des critères qui leur sont propres.
Il est possible d'observer des propriétés remarquables concernant le choix des profils fait précédemment. Par exemple, le troisième plus court chemin π_3 qui quitte la ligne 2 du métro pour le réutiliser à la fin de son parcours est un chemin tout à fait singulier pour le profil A qui y verra une disparité avec les deux premiers chemins alors que ce chemin sera considéré équivalent au deuxième chemin π_2 pour le profil B et équivalent au chemin π_1 pour le profil C.

chemin	nœuds	coût	modes	localisation	profil A	profil B	profil C
π_1	1-2-5-7	6	$s_2s_2s_2$	nnn	s_2	s	n
π_2	1-2-4-7	7	$s_2b_1b_1$	nnc	s_2b_1	bs	nc
π_3	1-2-4-5-7	8	$s_2b_1s_1s_2$	$nnnn$	$s_2b_1s_1s_2$	bs	n
π_4	1-4-7	8	pb_1	cc	pb_1	bp	c
π_5	1-2-4-6-7	9	$s_2b_1b_2b_2$	$nnss$	$s_2b_1b_2$	bs	ns
π_6	1-3-6-7	10	ppb_2	sss	pb_2	bp	s
π_7	1-3-4-7	11	ps_1b_1	ssc	ps_1b_1	bps	sc
π_8	1-3-4-5-7	12	$ps_1s_1s_2$	$ssnn$	ps_1s_2	ps	sn
π_9	1-3-4-6-7	13	$ps_1b_2b_2$	$ssss$	ps_1b_2	bps	s

TABLE 3.13 – Caractérisation des chemins élémentaires du graphe G_3 pour chacun des profils A, B et C pour un trajet du nœud 1 vers le nœud 7.

3.6.3 Algorithme

L'algorithme présenté dans le pseudo-code 7 permet, à partir d'un ensemble de chemins élémentaires P et d'une valeur de seuil t, de déterminer au plus k chemins alternatifs suivant une fonction de mesure de la différence d. Pour cela, l'ensemble des solutions S est initialisé avec le plus court chemin (ligne 2). Si pour un chemin élémentaire donné p', la différence suivant la fonction d avec tous les chemins déjà présents dans l'ensemble solution est supérieure à la valeur de seuil t (lignes 5 à 10) alors il est ajouté à la solution (ligne 12). Tous les chemins sont alors énumérés jusqu'à ce que le nombre k de chemins voulus soit atteint (ligne 15) ou que tous les chemins élémentaires disponibles aient été testés (ligne 18).
La complexité de cette méthode est de $O(k^2 \times O_{diff}(n))$ où $O_{diff}(n)$ est la complexité de l'algorithme choisi pour le calcul de la différence sur deux séquence de longueur n le nombre de nœuds du graphe.

Algorithme 7 Algorithme de sélection des alternatives

```
1:  function ALTERNATIVES(P, t, k, d)
2:      S ← {P[0]}
3:      Pour i ∈ [1, |P| − 1] faire
4:          (sélectionné, p) ← (true, P[i])
5:          Pour p′ ∈ S faire
6:              Si d(p′, p) < t alors
7:                  sélectionné = Faux
8:                  Fin de boucle
9:              Fin Si
10:         Fin Pour
11:         Si sélectionné alors
12:             S ← S ∪ {P[i]}
13:         Fin Si
14:         Si |S| = k alors
15:             retourner S
16:         Fin Si
17:     Fin Pour
18:     retourner S
19: Fin function
```

3.6.4 Méthodes de sélection

Nous nous intéressons dans cette partie à l'évaluation de deux méthodes de calcul de distance entre chemins décrits par une séquence caractéristique. Pour cela, deux méthodes classiques de la littérature ont été testées. La première repose sur la distance de Levenshtein alors que la seconde s'appuie sur la notion de n-grammes.

3.6.4.1 Distance de Levenshtein

La distance de Levenshtein est une mesure de la différence entre deux chaînes de caractères. Elle est définie comme étant le nombre minimal de modifications unitaires d'insertion ou de suppression d'une lettre pour transformer un mot source en un mot destination. Les méthodes classiques de la littérature (Meyers) consistent en des algorithmes dynamiques terminant en $O(N \times D)$ où D est la distance de Levenshtein et N la longueur du mot le plus long. La distance de Levenshtein entre deux chemins sera toujours un entier positif ou nul, et peut être utilisée comme seuil à spécifier pour l'algorithme de sélection des alternatives.
Prenons l'usager de profil A vu dans l'exemple de la partie 3.6.2. Au début, le chemin π_1 est retenu avec pour mot s_2. Ensuite il faut déterminer si le chemin π_2 va faire partie de la solution. Comme le mot qui lui est associé est s_2b_1, la différence calculée avec la distance de Levenshtein avec le chemin π_1 est de 1 puisqu'il suffit de rajouter au premier le suffixe b_1 pour obtenir le second. Si le seuil fixé est de zéro, c'est à dire qu'aucun filtrage n'est demandé, alors le chemin π_2 est retenu. Dans le cas où la valeur du seuil est de 1 alors le chemin est retenu. En revanche, pour toutes les valeurs de seuils supérieures à 1, le chemin est rejeté puisque sa différence par rapport aux chemins déjà sélectionnés n'est pas assez importante pour qu'il soit considéré comme pertinent.
Le tableau 3.14 établit pour chaque profil ainsi que pour différentes valeurs de seuil les chemins retenus par le filtrage associé.

Seuil	profil A	profil B	profil C
0	tous	tous	tous
1	tous	(π_1, π_2, π_4, π_7, π_8)	(π_1, π_2, π_4, π_5, π_6, π_7, π_8)
2	(π_1, π_3, π_4, π_5, π_6, π_8)	(π_1, π_4)	(π_1, π_4, π_6)
3	(π_1, π_3, π_4, π_9)	(π_1, π_4)	(π_1, π_7)

TABLE 3.14 – Chemins alternatifs sélectionnés par la distance de Levenshtein.

3.6.4.2 N-grammes

Les n-grammes sont des sous-séquences de longueur n d'une séquence plus large. Leur décompte dans une séquence permet de calculer la probabilité d'apparition d'une sous-séquence dans un corpus, mesurant ainsi une similarité.
L'utilisation d'un tel principe peut être reproduite dans notre cas par le décompte des

n-grammes d'un chemin. La comparaison entre deux chemins s'effectue ensuite par leur nombre d'occurences de n-grammes identiques divisé par la moyenne du nombre total de n-grammes des chemins (autrement dit multiplié par 2 et divisé par le nombre de n-grammes des chemins). Cette valeur est alors comprise entre 0 et 1 et plus le résultat obtenu est proche de 0 plus les chemins sont différents. Les valeurs de seuil possibles seront donc des réels compris entre 0 et 1 et plus la valeur sera petite plus la distinction entre les chemins sera forte. Pour se ramener à une mesure de différence, nous prendrons donc l'opposé des n-grammes comme mesure de différence entre deux chemins. Puisqu'il n'y a pas de n-grammes significatifs dans un mot à une lettre il convient d'ajouter la lettre vide ε en début et fin de chaque mot afin que ceux-ci comportent au moins trois lettres pour garantir que ces 2-grammes puissent fonctionner. La complexité temporelle d'une telle méthode est $O(N \times log(N))$

Reprenons à présent pour exemple le profil A. Les chemins π_1 et π_2 ont pour 2-grammes respectivements $\{(\varepsilon, s_2), (s_2, \varepsilon)\}$ et $\{(\varepsilon, s_2), (s_2, b_1), (b_1, \varepsilon)\}$. La valeur de similarité obtenue est donc de $-(1/2, 5) = -0,2$ et le deuxième chemin est rejeté pour tous les seuils supérieurs ou égaux à $-0,2$. Le chemin π_3 se décompose en $\{(\varepsilon, s_2), (s_2, b_1), (b_1, s_1), (s_1, s_2), (s_2, \varepsilon)\}$. Son écart avec π_1 est donc de $-(2/(7/2)) = -0,57$ et l'écart avec π_2 est de $-(2/4) = -0,5$. Le chemin π_3 sera donc rejeté si le seuil choisi est supérieur ou égal au maximum des deux valeurs c'est à dire à -0.5.

Le tableau 3.15 établit pour chaque profil ainsi que pour plusieurs valeurs de seuil les chemins retenus par le filtrage associé.

Seuil	profil A	profil B	profil C
-1	tous	tous	tous
-1/2	(π_1, π_2, π_4, π_6, π_7, π_8)	(π_1, π_2, π_4)	(π_1, π_2, π_4, π_5, π_6, π_7, π_8)
-1/3	(π_1, π_4, π_5, π_6, π_7)	(π_1, π_4)	(π_1, π_4, π_6)
0	(π_1, π_4)	(π_1, π_4)	(π_1, π_4, π_6)

TABLE 3.15 – Chemins alternatifs sélectionnés par la méthode des n-grammes.

3.6.5 Analyses expérimentales

Afin d'analyser la performance de ces deux méthodes pour sélectionner des plus courts chemins alternatifs la méthodologie des expérimentations présentées en section 3.5.1 est reprise. Le calculs de 500 chemins avec cycles a permis d'obtenir un nombre satisfaisant de chemins élementaires (en moyenne 490) sur un jeu de 40 instances. Ces chemins ont ensuite été triés et sélectionnés par la méthode des n-grammes et la distane de Levenshtein selon les trois profils suivants :

- le profil D associe à chaque arc son type de transport (bus, métro, piéton, etc) ;
- le profil E écrit une unique fois le mode de toute séquence d'arcs consécutifs appartenant à un même mode ;

- le profil F procède de la même façon que le profil E mais se base non seulement sur les modes mais également sur les différentes lignes composant ces modes lorsque cela est possible.

Les profils A, B et surtout C n'ont pas été implémentés mais sont néanmoins conservés pour leur simplicité.
Cette méthode de génération de chemins puis de leur tri ne correspond pas à la méthodologie décrite où les chemins sont générés par l'algorithme de calcul de k plus courts chemins et analysés au fur et à mesure. Cela est fait afin de pouvoir comparer à quantité de chemins égale les performances des différentes méthodes de sélection des alternatives.

3.6.6 N-grammes

Le filtrage utilisant les n-grammes prend en moyenne 500 millisecondes pour s'exécuter mais donne de très mauvais résultats quels que soient les profils des usagers. Pour un seuil de 2 il y a en moyenne 1.85 et 2.28 chemins pour des n-grammes de longueur 2 et 3 respectivement. Ensuite, pour un seuil de 3 ou plus le nombre de chemins tombe en dessous de 1.5 avec toujours une préférence pour le n-gramme le plus long.

3.6.7 Distance de Levenshtein

Le tableau 3.16 synthétise l'ensemble des résultats expérimentaux obtenus pour un filtrage utilisant la distance de Levenshtein pour des seuils allant de 0 jusqu'à 5. Avec un seuil de 0, c'est à dire sans aucun filtre, le temps d'exécution maximal est de 137 millisecondes et les temps d'exécution pour les autres seuils y sont inférieurs. De plus, il est normal d'obtenir pour un seuil de zéro le même nombre de chemins pour chaque profil puisqu'alors aucun filtrage n'est actif. Lorsque la valeur du seuil augmente des différences flagrantes apparaissent entre les différents profils. Le profil D correspond à un cas idéal pour lequel un nombre de 5.56 chemins alternatifs sont trouvés en moyenne tout en diminuant légèrement pour des valeurs de seuil plus grand. En revanche le profil N n'est pas capable de faire une différence entre les chemins, il n'est pas assez sélectif puisque le nombre de chemins filtrés s'établit tout de suite à 1 dès que le seuil prend une valeur significative. A l'inverse le profil F conserve trop de chemins pour qu'il soit utilisable et n'arrive jamais à filtrer des chemins.

seuil	D	E	F
0	490.86	490.86	490.86
1	5.56	1.017	490.86
2	3.01	1.014	433.52
3	2.20	1.010	414.04
4	1.79	1.009	330.45
5	1.57	1.001	324.29

TABLE 3.16 – Résultats expérimentaux basés sur la distance de Levenshtein pour les différents utilisateurs.

3.7 Conclusion

Nous avons présenté dans ce chapitre des adaptations d'algorithmes de la littérature au cas spécifique du calcul de k plus courts chemins dans des graphes de transport suivant deux volets : le calcul de chemins élémentaires et celui de chemins contenant des cycles. Cependant, l'introduction de la dépendance au temps ne permet plus l'adaptation de certains algorithmes.

L'objectif global de l'approche est de calculer des alternatives au plus court chemin en se basant sur une liste de chemins élémentaires triés par ordre croissant de coût et différenciés par une fonction qui est potentiellement inconnue ou dont la définition demande des compétences en transport et en aménagement du territoire. Pour l'algorithme REA qui a été étendu jusqu'à ce point, le nombre de chemins élémentaires obtenus n'est pas compatible avec une telle approche.

C'est pour palier ce problème que nous proposons l'algorithme IEA qui peut supprimer les cycles de longueur donnée au cours de la recherche. Le développement et les expérimentations de cet algorithme ont montré que le temps de calcul pour calculer un nombre important de chemins élémentaires est bien moindre que celui que requièrent les algorithmes de la littérature cherchant des chemins élémentaires dès le départ. Toutefois, si l'algorithme est heuristique, les coûts des chemins obtenus par l'application de l'algorithme IEA sont comparativement proches de ceux obtenus par l'algorithme de Yen si bien que l'approche reste intéressante. Et cet aspect de calcul de chemins élémentaires par le biais d'algorithmes calculant des chemins avec cycles n'a à notre connaissance pas été étudié dans la littérature alors que nous avons montré qu'elle a du sens expérimentalement. Cependant, l'algorithme est vite limité par la taille mémoire qu'il nécessite puisqu'un tas est stocké pour chacun des nœuds du graphe. Enfin, il est peu probable que la méthode fonctionne bien sur des graphes dont la topologie est différente de celle des réseaux de transport. En particulier, si les chemins recherchés contiennent beaucoup de sommets, la longueur des cycles qui vont apparaître rapidement va vite dépasser la longueur des cycles supprimés rendant alors la méthode inopérante.

En plus de fournir des entrées à un algorithme de calcul de plus courts chemins alternatifs, le calculs de k plus courts chemins peut être utilisé dans l'analyse de réseau afin d'identifier les zones mal desservies. Plus il y a de chemins qui ont des topologies différentes et des coûts voisins entre deux nœuds et plus l'on peut qualifier l'offre de transport d'efficace.

L'algorithme IEA et le calcul d'alternatives générique peuvent donc être employés à double escients.

Chapitre 4

Synchronisation de trajets : application au covoiturage

Le covoiturage consiste à mettre en relation des personnes sollicitant une offre de transport pouvant être assurée au moins partiellement par d'autres utilisateurs lorsque ces derniers effectuent tout ou partie de leur trajet en véhicule personnel. Ces rapprochements se déclinent souvent en deux concepts dont la résolution peut être concomitante, à savoir l'appariement des usagers entre eux et le calcul des itinéraires pour chacun.
Nous nous intéressons dans ce chapitre à un problème de synchronisation d'itinéraires pour deux usagers en nous focalisant sur le calcul d'itinéraires, supposant que l'appariement a déjà été décidé par ailleurs. Un conducteur dispose d'un véhicule pour effectuer un trajet durant lequel il peut prendre en charge le passager pouvant utiliser les transports en commun pour une partie de son itinéraire. Ce problème, appelé `2SPSPP` («2-Synchronization Points Shortest Path Problem» ou Problème de Plus court Chemin à 2 Points de Synchronisation) dans [26], consiste à déterminer un point de rencontre et un point de dépose de telle sorte que la somme des horaires d'arrivée à destination des deux usagers soient minimisée.
Ce chapitre débute par la définition du problème étudié (`2SPSPP`) dans la section 4.1. Puis nous considérons différentes variantes du problème `2SPSPP` dans la section 4.2. Enfin, la section 4.3 présente des résultats expérimentaux sur certaines des extensions considérées.

4.1 Position du problème

4.1.1 Le problème `2SPSPP`

Afin d'améliorer l'efficacité des systèmes de covoiturage, il a été montré dans [14] l'intérêt de pouvoir prendre en charge ou déposer un passager en divers points du réseau de transport et non pas seulement aux origines ou destinations des trajets des participants. Il est donc nécessaire de déterminer des points de rencontre de bonne qualité pour les deux usagers (conducteur et passager). Ainsi, pour deux usagers donnés, à partir de leurs origines et destinations respectives et d'un horaire de départ pour chacun, il convient de déterminer leurs points de rencontre et de séparation ainsi que les trajets associés (des origines vers le point de rencontre, du point de rencontre vers le point de séparation et du point de séparation vers les destinations) de telle sorte que les horaires

à destination soit minimisés. Ce problème, appelé 2SPSPP, a été introduit dans [26]. La figure 4.1 schématise un cas dans lequel un conducteur c et un piéton p avec leurs origines et destinations respectives (o_v, d_v) et (o_p, d_p) et leurs horaires de départ τ_v et τ_p cherchent à synchroniser leurs trajets et donc à déterminer un point de rencontre (pick-up), noté r, et un point de séparation (drop-off), noté s de telle sorte que la somme de leurs horaires d'arrivée soit minimisée. Chaque solution du problème de covoiturage est donc composée de cinq trajets $(o_p - r,\ o_v - r,\ r - s,\ s - d_p,\ s - d_v)$. Le piéton ne peut se déplacer qu'à pied ou en transport en commun (ces modes sont dénotés par $\{m, tc\}$) et le conducteur ne peut se déplacer qu'en voiture (le mode est dénoté $\{v\}$). Les flèches pointillées bleues représentent les trajets qui dépendent du temps.

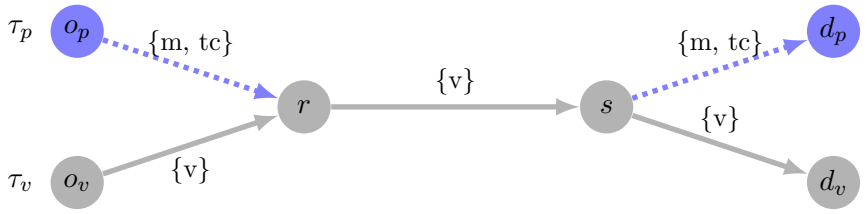


FIGURE 4.1 – Problème 2SPSPP

Dans le 2SPSPP, les modes de déplacement à pieds ou en voiture sont supposés statiques et le mode de déplacement en transport en commun est supposé dépendant du temps (en raison de l'existence de tables horaires ou de lignes à fréquence).

4.1.2 Méthode de résolution

Le problème a été montré polynomial et peut être résolu par une méthode énumérant l'ensemble des couples de pick-up et de drop-off d'un réseau de transport (soit n^2 couples pour un réseau de n sommets) et en calculant pour chacun d'entre eux les valeurs des cinq plus courts chemins composant une solution de covoiturage. La prise en compte des horaires se fait en débutant par le calcul des deux premiers algorithmes vers le point de rencontre où l'horaire maximal est alors connu. Puis d'un autre entre ce point et le point de destination et enfin de deux algorithmes vers les destinations respectives du piéton et de la voiture.
Pour éviter cette méthode énumérative, inefficace sur des réseaux de taille réelle, une autre approche de résolution a été proposée. Cette approche illustrée sur la figure 4.2 se base sur 5 algorithmes de plus courts chemins entrelacés. Le but est d'éviter le calcul de plus courts chemins en parcours arrière dans les graphes dépendant du temps. En effet, si tel était le cas, il faudrait effectuer les calculs sur des valeurs de parcours minimales afin d'obtenir une borne sur le coût global du trajet. Cette borne serait alors réutilisée lors d'un parcours supplémentaire afin d'en limiter l'espace de recherche.

Initialement, deux algorithmes de calcul de plus courts chemins, A_1 et A_2 démarrent leurs explorations en forward depuis les origines o_v et o_d, et un algorithme de calcul

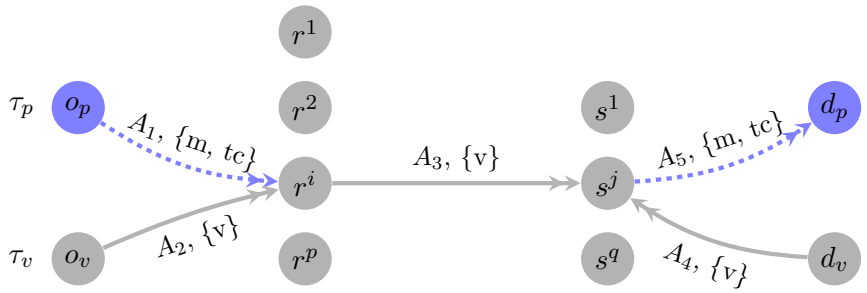


FIGURE 4.2 – Principe de résolution du 2SPSPP

de plus court chemin A_4 débute en backward depuis la destination du conducteur d_v. L'algorithme A_1 est un algorithme basé sur les modes $\{m, tc\}$, les temps de trajet sont donc dépendants du temps, alors que les algorithmes A_2 et A_4 utilisent uniquement le mode $\{v\}$ pour lequel les temps de trajet sont statiques. Lorsqu'un sommet est atteint par les algorithmes A_1 et A_2 il devient un point de rencontre r potentiel et l'algorithme A_3 peut débuter son exploration en forward. A_3 se base uniquement sur le mode $\{v\}$ en statique. Lorsqu'un sommet est atteint par les algorithmes A_3 et A_4 il devient un point de séparation s potentiel et l'algorithme A_5 débute son exploration en forward vers la destination du piéton d_p. A_5 considère les modes $\{m, tc\}$ et dépend du temps. A chaque étape de résolution, un seul de ces 5 algorithmes est actif. De cette façon le principe de programmation dynamique consistant en la fixation d'étiquettes est respecté permettant de résoudre le problème de façon optimale. La méthode s'arrête lorsque la destination du piéton d_p est marquée par l'algorithme A_5.

En pratique, les algorithmes A_1, A_2, A_3 et A_4 sont des algorithmes de Dijkstra (utilisant des coûts dépendant des horaires pour l'algorithme A_1). Si des contraintes sont explicitées sur l'utilisation des modes de transport, que ce soit pour le piéton ou pour le conducteur, l'algorithme de Dijkstra correspondant est remplacé par un algorithme de type DRegLC. L'algorithme A_5 correspond à un problème de plus court chemin spécifique, appelé *meilleure origine* («*Best Origin*») dans [26], et visant à déterminer quel sommet origine conduit au plus court chemin à destination. Lorsque les coûts dépendent des horaires, il a été montré dans [26] que le problème *meilleure origine* se résolvait à l'aide d'un algorithme de type plus court chemin pour lequel plusieurs labels doivent être conservés et propagés à chaque sommet, comme dans le cas des algorithmes de plus courts chemins multi-objectifs.

Dans cet algorithme A_5, une règle de dominance permet d'élaguer les labels non dominés afin de restreindre l'exploration. Deux règles de dominance ont été considérées et évaluées dans [26] : une règle dite exacte garantissant l'obtention de la solution optimale et une règle dite heuristique pouvant éliminer des labels conduisant à la solution optimale. Pour un nœud x, si on considère deux étiquettes $l_x = (\pi_x, \tau_x)$ et $l'_x = (\pi'_x, \tau'_x)$, où π_x et π'_x sont les coûts et τ_x et τ'_x sont les horaires, alors le label l_x domine le label

l'_x, i.e. $l_x \prec l'_x$, si et seulement si

- $\tau_x \leq \tau'_x \ \wedge \ \pi_x \leq \pi'_x$ pour la dominance heuristique ;
- $\tau_x \leq \tau'_x \ \wedge \ \pi_x - \pi'_x \leq \tau_x - \tau'_x$ pour la dominance exacte.

Il est à relever que la règle de dominance exacte ne marche que dans le cas où le coût correspond au temps. En effet, dans la preuve, il est supposé que les temps et les coûts sont homogènes puisqu'ils sont comparés, additionnés ou soustraits. Or ils pourraient être dans le cas général non homogènes c'est à dire dans des unités différentes ou de grandeurs différentes et non comparables.
A la fin de la méthode, on obtient les points de rencontre et de séparation pour lesquels la somme des heures d'arrivée est minimisée. En revanche, il y a en général un temps d'attente au point de rencontre pour l'un ou l'autre des participants. Cette attente pourrait être réduite à zéro dans le cas d'un trajet sans dépendance au temps et pourrait être minimisée lorsque ce dernier dépend du temps. Il faudrait alors recalculer un trajet avec un départ au plus tard pour arriver au point de rencontre à l'heure prévue pour le rendez-vous. Ainsi, le coût de la solution peut être légérement inférieur au coût calculé bien que cela n'affecte pas les heures d'arrivée.

La complexité de l'approche proposée est $O(4 \times \mathcal{C}_D + \mathcal{C}_{D_{ML}})$ où $\mathcal{C}_D$ est la complexité d'un algorithme de résolvant le problème de plus court chemin (Dijkstra ou DRegLC), et $\mathcal{C}_{D_{ML}}$ celle d'un algorithme résolvant le problème *meilleure origine* (Dijkstra multi-labels pour le cas où les coûts dépendent des horaires).
Les expérimentations effectuées sur le réseau constitué des régions Aquitaine et Midi-Pyrénées contenant des données de transport en commun pour les agglomérations de Bordeaux et Toulouse consistaient à effectuer un covoiturage au départ de Bordeaux, amenant le piéton sur Toulouse alors que la voiture finissait son trajet à Albi. Les résultats obtenus ont montré que la perte de qualité de la solution en utilisant la règle de dominance heuristique était très faible (de l'ordre de 0.05%) alors que les gains en temps de calcul étaient d'un ordre de grandeur environ.

Le problème `2SPSPP` a été étendu à la prise en compte d'une limite sur le temps maximal de trajet pour le piéton avant sa prise en charge par un conducteur depuis son origine. Pour cela l'algorithme A_1 est stoppé avec cette valeur limite et l'algorithme A_2 se base sur une exploration guidée vers la zone atteignable par le piéton. La prise en compte de cette contrainte sur le temps de trajet initial du piéton permet d'avoir des méthodes de résolution encore plus efficaces en termes de temps de calcul car cela restreint l'espace des points de rencontre possible. Cette contrainte peut être justifiée en pratique en considérant qu'un piéton qui ne peut être pris en charge rapidement par un véhicule utilisera son propre véhicule pour se déplacer.

4.1.3 Proposition d'un algorithme générique pour le 2SPSPP

Le pseudo-code 8 présente une version générique de l'algorithme 2SPSPP se basant sur l'utilisation de fonctions abstraites définies dans le pseudo-code 9. Pour chacun des 5 algorithmes i, les ensembles H_x^i stockent pour chaque nœud x un quadruplet (c, τ, r, s) où c et τ sont respectivement le coût et le temps courant sur le nœud quand r et s sont respectivement le point de rencontre et de séparation associés au parcours jusqu'à x, s'ils existent. De plus, les fonctions f et g fournissent le coût et le temps de traversée des arcs. Chacune prend donc comme premier paramètre l'arc à parcourir et comme second l'heure à laquelle il est parcouru. Pour le cas étudié, la fonction f utilisée est la fonction f^{ea} qui fournit le coût d'un arc parcouru dans le sens avant et la fonction g utilisée est la fonction g^{ea} qui fournit l'horaire d'arrivée suite à la traversée d'un arc à une heure donnée.
Au début, les coûts et temps sont initialisés à l'infini. Puis les nœuds d'origine du conducteur o_v et du piéton o_p sont initialisés avec leurs horaires respectifs τ_v et τ_p. De même pour la destination du conducteur d_v mais sans horaire donné puisque le trajet n'est pas dépendant du temps. Ces trois étiquettes sont ensuite empilées dans le tas H et tant que celui-ci n'est pas vide, l'algorithme effectue les opérations suivantes.
La meilleure étiquette correspondant à un nœud non encore traité de l'un des 5 algorithmes est récupérée (ligne 8). S'il s'agit du nœud destination du piéton d_p pour l'algorithme 5 alors la solution est retournée (ligne 11). Dans le cas inverse, les successeurs sont mis à jour les uns après les autres (lignes 13 à 22). Si l'algorithme courant est différent du cinquième alors la dominance s'établit sur le coût. En revanche, pour l'algorithme A_5 une des règles de dominance, exacte ou heuristique, permet de savoir si l'extension à partir du candidat est valide (ligne 17). Mais cette mise à jour n'a lieu qu'à la condition que le nœud suivant ne soit pas filtré par la fonction Filtre. Cette fonction, qui n'est pas utile dans la version simple de l'algorithme permet des extensions pouvant restreindre l'espace d'exploration de chacun des sommets de chaque algorithme comme c'est le cas pour l'extension proposée dans le dernier paragraphe de la section 4.1.3.
Enfin, si les algorithmes A_1 et A_2 ont tous deux fixé leur étiquette sur le sommet dépilé, alors ce nœud est empilé dans l'algorithme A_3, de même avec les algorithmes A_3 et A_4 (lignes 23 à 29 et 30 à 36). Pour cela, les fonctions Rencontre et Séparation déterminent si l'insertion a lieu et avec quels coût et horaire.

Pour la version originelle de l'algorithme 2SPSPP, les fonctions décrites dans le pseudo-code 9 sont utilisées. La fonction de parcours Itère utilisée est celle du parcours avant Suivant. Un filtrage peut-être appliqué, par exemple en fixant une limite sur le temps du trajet effectué seul par le piéton, la recherche des successeurs d'un nœud se fait vers l'avant et la règle de dominance utilisée peut être soit exacte, soit heuristique. Enfin, les fonctions Rencontre et Séparation déterminent les coûts et les horaires aux points de rencontre et de séparation. Pour un point de rencontre, le coût correspond à la somme des coûts des utilisateurs auquel s'ajoute un éventuel temps d'attente, l'horaire

Algorithme 8 Algorithme générique de résolution du 2SPSPP

function 2SPSPP($G = (V, E)$, $o_v \in V$, $o_p \in V$, $d_v \in V$, $d_p \in V$, $\tau_v \in \mathbb{R}$, $\tau_p \in \mathbb{R}$, $f : \mathbb{R}_+ \times E \to \mathbb{R}_+$, $g : \mathbb{R}_+ \times E \to \mathbb{R}_+$, FILTRE, ITÈRE, DOMINE, RENCONTRE, SÉPARATION)
  $H_x^i \leftarrow (+\infty, +\infty, \emptyset, \emptyset) \quad \forall i \in [1,5], \quad \forall x \in V$
  $H_{o_p}^1 \leftarrow (0, \tau_p, -, -)$
  $H_{o_v}^2 \leftarrow (0, \tau_v, -, -)$
  $H_{d_v}^4 \leftarrow (0, -, -, -)$
  $H \leftarrow \{H_{o_p}^1, H_{o_v}^2, H_{d_v}^4\}$
  Tant que $H \neq \emptyset$ **faire**
    $h_x^k = (c, \tau, r, s) \leftarrow min_c(H = \{(c, -, -, -)\})$
    $H \leftarrow H \setminus \{h_x^k\}$
    Si $x = d_p \wedge k = 5$ **alors**
      retourner h_x^k
    Fin Si
    Pour tout $(y, e) \in$ ITÈRE(G, x) **faire**
      $(c_{ref}, \tau_{ref}, -, -) \leftarrow H_y^k$
      $(c^+, \tau^+) \leftarrow (c + f(e, c), \tau + g(e, \tau))$
      Si $\neg$FILTRE$(k, y, \{H^i\}_{i\in[1,5]})$ **alors**
        Si $(c^+ < c_{ref} \vee (k = 5 \wedge$ DOMINE$((c^+, \tau^+), (c_{ref}, \tau_{ref}))))$ **alors**
          $H \leftarrow H \setminus \{H_x^k\}$
          $H_x^k \leftarrow (c^+, \tau^+, r, s)$
          $H \leftarrow H \cup \{H_x^k\}$
        Fin Si
      Fin Si
      Si $(k = 1 \vee k = 2) \wedge H_x^1[0] \neq +\infty \wedge H_x^2[0] \neq +\infty$ **alors**
        $(pick, c^+, h^+) \leftarrow$ RENCONTRE$(x, \{H^i\}_{i\in[1,5]})$
        Si *pick* **alors**
          $H_x^3 \leftarrow (c^+, h^+, x, -)$
          $H \leftarrow H \cup \{H_x^3\}$
        Fin Si
      Fin Si
      Si $(k = 3 \vee k = 4) \wedge H_x^3[0] \neq +\infty \wedge H_x^4[0] \neq +\infty$ **alors**
        $(drop, c^+, h^+) \leftarrow$ SÉPARATION$(x, \{H^i\}_{i\in[1,5]})$
        Si *drop* **alors**
          $H_x^5 \leftarrow (c^+, h^+, r, x)$
          $H \leftarrow H \cup \{H_x^5\}$
        Fin Si
      Fin Si
    Fin Pour
  Fin Tant que
  retourner $\emptyset$
Fin function

correspond au maximum des horaires des deux usagers. Pour un point de séparation, le coût correspond à la somme des coûts des utilisateurs et l'horaire est fourni directement par l'algorithme A_3 (heure d'arrivée au point de séparation).

Algorithme 9 Fonctions pour l'algorithme 2SPSPP

function SUIVANT($G = (V, E), x \in V$)
    retourner $\{(y, (x, y)) \mid (x, y) \in E\}$
Fin function

function FILTRE(k, x, $\{H^i\}_{i\in[1,5]}$)
    retourner Faux
Fin function

function DOMINE HEURISTIQUEMENT((c^+, τ^+), (c_{ref}, τ_{ref}))
    retourner $c^+ < c_{ref} \wedge \tau^+ < \tau_{ref}$
Fin function

function DOMINE EXACTEMENT((c^+, τ^+), (c_{ref}, τ_{ref}))
    retourner $c^+ < c_{ref} \wedge (c^+ - c_{ref}) < (\tau^+ - \tau_{ref})$
Fin function

function RENCONTRE(x, $\{H^i\}_{i\in[1,5]}$)
    $(c_1, \tau_1, -, -) \leftarrow H_x^1$
    $(c_2, \tau_2, -, -) \leftarrow H_x^2$
    retourner $(\textbf{Vrai}, c_1 + c_2 + |\tau_1 - \tau_2|, max(\tau_1, \tau_2))$
Fin function

function SÉPARATION(x, $\{H^i\}_{i\in[1,5]}$)
    $(c_3, \tau_3, -, -) \leftarrow H_x^3$
    $(c_4, \tau_4, -, -) \leftarrow H_x^4$
    retourner $(\textbf{Vrai}, c_3 + c_4, \tau_3)$
Fin function

4.2 Etude de variantes du 2SPSPP

Le problème 2SPSPP est assez général car il ne présuppose rien sur le type de parcours de chacun des utilisateurs. Cependant, la considération de cas particuliers peut simplifier ou complexifier le problème. Nous proposons dans cette partie de fournir une liste non exhaustive de différentes variantes en détaillant les changements induits dans la complexité du problème ainsi que les modifications qu'il faut effectuer dans l'algorithme originel pour résoudre le problème.

4.2.1 Origines ou destinations identiques

Nous considérons dans cette partie les sous-problèmes du 2SPSPP dans lesquels les origines ou les destinations des utilisateurs sont identiques. Ces deux cas sont illustrés dans les figures 4.3 et 4.4 et se ramènent à rechercher un seul point de synchronisation.

Comme pour le 2SPSPP, seuls les durées de trajet du piéton dépendent des horaires et les horaires sont connus depuis les origines des usagers. Les problèmes ainsi caractérisés peuvent être appelés 1SPSPP et peuvent se résoudre en considérant seulement trois des cinq algorithmes nécessaires à la résolution du 2SPSPP. Néanmoins, selon la situation (origine ou destination identique), l'utilisation d'algorithmes dans le sens backward et/ou la présence de trajets dépendant des horaires entrainent ou non des difficultés. Lorsque les deux usagers ont une même destination, trois algorithmes dans le sens forward suffisent pour résoudre le 1SPSPP, comme illustré sur la figure 4.3. Si l'on considère en plus que seul le piéton utilise des modes de transport dépendant du temps, la complexité de l'approche est $O(3 \times \mathcal{C}_D)$.

Principes de résolution du 1SPSPP pour des

FIGURE 4.3 – destinations communes

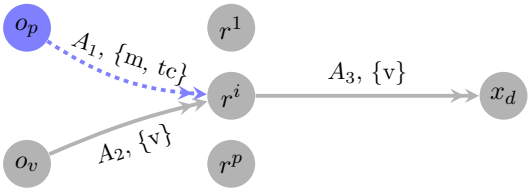
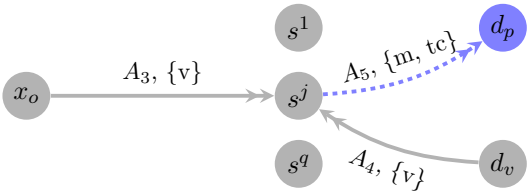


FIGURE 4.4 – origines communes



Cependant, lorsque c'est l'origine qui est commune aux deux utilisateurs, avec l'hypothèse que le trajet du piéton est dépendant des horaires, un algorithme multi-label devient nécessaire depuis le point de séparation jusqu'à la destination du piéton pour le calcul A_5 (voir figure 4.4). L'approche de résolution s'appuie sur un algorithme dans le sens forward depuis l'unique origine, un algorithme dans le sens backward depuis la destination du conducteur et un algorithme multi-label depuis le point de séparation, sa complexité est donc $O(2 \times \mathcal{C}_D + \mathcal{C}_{D_{ML}})$.

4.2.2 Impact de la dépendance au temps

Une des hypothèses faites dans [26] est que les arcs de voirie ont des temps de parcours constants. Nous étudions dans cette partie l'ensemble des variantes liées à la dépendance au temps ou non des différents trajets du piéton et du conducteur.

4.2.2.1 Dépendance au temps des trajets du piéton

Dans l'approche développée pour le 2SPSPP, le trajet du piéton est composé de deux sous-trajets. Chacun d'entre eux est déterminé par un algorithme spécifique : l'algorithme A_1 pour le trajet o_p – r, et l'algorithme A_5 pour le trajet s – d_p qui est un algorithme multi-label. Nous allons considérer dans cette partie l'impact de la dépendance au temps pour chacun de ces trajets.

1. Si les trajets du piéton sont tous indépendants du temps (par exemple s'il ne prend pas les transports en commun), alors les cinq algorithmes utilisés sont tous des algorithmes de type Dijkstra (car il a été montré dans [26] qu'un tel agorithme résolvait le problème Best Origin dans ce contexte), la complexité de l'approche se ramène alors à celle d'un algorithme de Dijkstra : $O(5 \times \mathcal{C}_D)$.

2. Supposons que le coût des arcs soit constant sur la partie du trajet piéton entre son origine et le point de rencontre (trajet $o_p - r$) mais dépendant du temps depuis le point de séparation jusqu'à sa destination (trajet $s - d_p$). Ce cas de figure correspond au cas où le piéton n'utilise que la marche ou un véhicule particulier (voiture, vélo, ...) de son origine au point de rencontre. L'algorithme A_1 correspond alors à un algorithme de Dijkstra et l'algorithme multi-label, utilisé en A_5 pour résoudre le problème de la *meilleure origine* est toujours nécessaire. La complexité de l'approche est alors $O(4 \times \mathcal{C}_D + \mathcal{C}_{D_{ML}})$.

3. Supposons que les temps de trajet du piéton sont constants depuis le point de séparation jusqu'à sa destination et qu'ils sont dépendant du temps depuis son origine jusqu'au point de rencontre. Ce cas peut se produire par exemple si l'usager récupère un véhicule au point de séparation pour se rendre à sa destination, ou s'il effectue ce trajet là à pieds uniquement. L'algorithme A_1 reste un algorithme classique de plus court chemin (Dijkstra ou DRegLC), en revanche l'algorithme multi-label utilisé en A_5 peut être remplacé par un algorithme de Dijkstra. La complexité de l'approche se ramène alors à $O(5 \times \mathcal{C}_D)$.

4.2.2.2 Dépendance au temps des trajets du conducteur

Dans l'approche développée pour le 2SPSPP, le trajet du conducteur est composé de trois sous-trajets et chacun d'entre eux est déterminé par un algorithme différent (A_2 pour le trajet $o_v - r$, A_3 pour le trajet $r - s$ et A_4 pour le trajet $s - d_v$). Nous allons étudier dans cette partie l'impact de la dépendance au temps pour chacun de ces trajets.

1. Considérons le trajet $o_v - r$ et supposons qu'il soit dépendant du temps. Un tel cas peut se produire lorsque le temps de parcours du conducteur prend en compte l'état de la circulation au moment où il traverse chaque arc sur le trajet. Le cas similaire arriverait si les deux usagers partaient en transport en commun pour se retrouver à un endroit d'où ils peuvent partir avec un véhicule, comme ce pourrait être le cas d'un véhicule en libre partage. L'algorithme A_2 doit alors être modifié pour correspondre à l'algorithme DRegLC si des contraintes sur les modes de déplacement sont explicitées mais la dépendance au temps n'introduit pas d'inconsistance entre les coûts et les temps de trajet au point de rencontre. Les autres algorithmes de la méthode ne sont pas modifiés et donc la complexité de l'approche reste inchangée : $O(4 \times \mathcal{C}_D + \mathcal{C}_{D_{ML}})$.

2. Supposons maintenant que seul le trajet effectué en commun par les deux usagers, $r - s$, soit dépendant du temps. Comme il n'y a pas d'inconsistance au point de rencontre en provenance des algorithmes A_1 et A_2, la dépendance au temps sur la portion commune est traitée par l'algorithme A_3 qui reste un algorithme de type Dijkstra dépendant du temps. L'approche développée pour le `2SPSPP` reste applicable et la complexité est inchangée.
3. Si l'on suppose que cette dépendance au temps se produit uniquement sur la fin du trajet du conducteur, c'est à dire sur le trajet $s - d_v$, par exemple parce que le véhicule est laissé au point de séparation et que les deux usagers finissent à pieds et en transport en commun, alors la résolution développée pour le `2SPSPP` n'est plus applicable. En effet, l'application d'un algorithme de Dijkstra pour l'algorithme A_4 sur le graphe inverse en partant de la destination du conducteur n'est plus valable sans effectuer de recalcul puisque l'heure au point d'arrivée du conducteur n'est pas connue et que son trajet dépend du temps. La complexité du problème reste donc polynomiale par l'énumération de tous les couples mais cette approche naïve serait prohibitive.
4. Si les trois hypothèses sont retenues, alors la complexité est identique à celle faisant intervenir la dépendance au temps sur la dernière partie de trajet de la voiture et l'approche développée pour le `2SPSPP` n'est plus utilisable.

4.2.2.3 Conclusion

Dans cette section, l'ensemble des variantes supprimant ou introduisant des dépendances au temps, par rapport au problème `2SPSPP` initial, ont été prises en compte. Il en résulte que tant que les deux trajets vers les destinations ($s - d_p$ et $s - d_v$) ne sont pas tous les deux dépendants du temps, alors l'approche présentée dans [26] peut être utilisée avec une complexité de $O(4 \times \mathcal{C}_D + \mathcal{C}_{D_{ML}})$ dans le pire cas. Si les deux trajets $s - d_p$ et $s - d_v$ sont indépendants du temps, alors l'approche du `2SPSPP` peut se simplifier (en utilisant un algorithme de Dijkstra pour résoudre le problème de la *meilleure origine* de $s - d_p$), ce qui donne une complexité totale égale à celle d'un algorithme de Dijkstra.

4.2.3 Plusieurs passagers avec points de synchronisation identiques

Dans cette partie, nous considérons le cas où plusieurs passagers utilisent le même covoiturage en se synchronisant sur un point de rencontre et un point de séparation communs à tous. Nous notons par p le nombre de passagers et nous considérons que ce nombre est inférieur au nombre de places disponibles dans le véhicule de covoiturage. Nous supposerons, sauf mention spécifique, que seuls les trajets des piétons dépendent du temps et que les horaires de départ sont connus pour tous les usagers à partir de leurs origines.

Ce problème est polynomial, en effet il peut se résoudre par une méthode exacte de type brute-force en énumérant toutes les paires de points de rencontre et de séparation et en calculant les différents trajets des utilisateurs pour chacune de ces paires (en partant des origines pour connaitre les horaires), pour une complexité de $O((2 \times p + 3) \times V^2 \times \mathcal{C}_D)$. Nous allons considérer par la suite différentes situations permettant dans certains cas d'adapter l'approche développée pour résoudre le 2SPSPP.

1. Le cas où les passagers ont tous une même origine et une même destination est trivial et se résout directement avec l'approche proposée pour le 2SPSPP.

2. Lorsque les passagers ont tous une même destination, le problème se résout en utilisant pour chacun d'eux un algorithme de calcul de plus court chemin (Dijkstra ou DRegLC) depuis leurs origines respectives (ces algorithmes sont notés A_1^1 à A_1^p), puis lorsqu'un sommet est marqué par l'ensemble de ces algorithmes ainsi que par l'algorithme A_2 utilisé pour le trajet effectué par le conducteur depuis son origine, la suite de la résolution du problème se déroule comme celle du 2SPSPP.

3. Si les p passagers partagent une unique origine mais que leurs destinations sont différentes, se pose le problème du calcul des trajets du point de séparation à chaque destination. En effet, si ces différents trajets sont dépendants du temps, l'approche 2SPSPP ne peut plus être utilisée. En effet, l'algorithme pour résoudre le problème *meilleure origine* ne s'applique pas pour déterminer la meilleur origine pour plusieurs destinations lorsque l'ensemble des trajets dépend du temps. Au plus un seul trajet final peut dépendre du temps si l'on souhaite conserver la même méthode que pour résoudre le 2SPSPP.

4. Lorsque les piétons ont des origines et des destinations différentes et se synchronisent sur un point de rencontre et un point séparation, les cas présentés ci-dessus se combinent. La difficulté principale réside dans les calculs des trajets depuis le point de séparation jusqu'aux différentes destinations. Si tous les piétons ont des trajets indépendants du temps, il est possible de remplacer l'algorithme A_5 par un ensemble d'algorithmes en backward (ou en forward) et donc de généraliser l'approche 2SPSPP. Si un seul des piétons a un trajet dépendant du temps, l'approche 2SPSPP peut également s'appliquer. Si tous les trajets des piétons dépendent du temps, l'approche 2SPSPP n'est plus valable.

4.2.4 Horaires de départ ou d'arrivée

Dans le problème du 2SPSPP initial, les heures sont données aux origines des deux usagers ce qui revient à résoudre deux problèmes $EA_{o,d}(t)$, un pour le piéton et l'autre pour le conducteur. Nous considérons dans cette partie les extensions faisant varier la connaissance de l'heure de départ ou d'arrivée de chacun des participants.

4.2.4.1 2SPSPP avec heures d'arrivée

Dans la variante de départ au plus tard, $LD_{o,d}(t)$, du 2SPSPP, les deux passagers veulent arriver chacun à une heure donnée à leur destination. On peut appliquer l'approche proposée pour le 2SPSPP en inversant le sens de parcours de chacun des algorithmes comme illustré sur la figure 4.5.

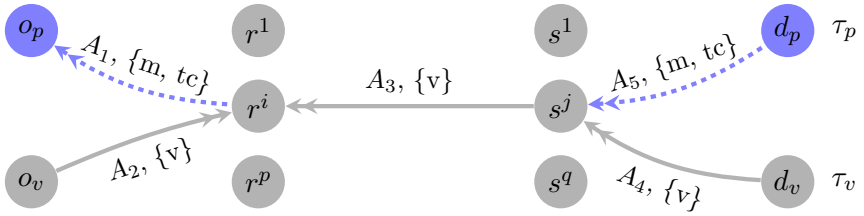


FIGURE 4.5 – Principe de résolution du 2SPSPP pour des départs au plus tard

Initialement, les algorithmes A_5 et A_4 sont initialisés avec les points de destination et effectuent un parcours du graphe en cheminement arrière (ce qui n'est pas problématique pour le calcul de A_5 car on connait l'heure d'arrivée à la destination τ_p). De plus, l'algorithme A_2 est initialisé avec l'origine du conducteur et effectue un parcours en avant du graphe. Lorsqu'un même nœud est marqué par les algorithmes A_5 et A_4, un point de séparation potentiel est détecté. Son étiquette peut être insérée pour l'algorithme A_3 qui détermine le trajet commun en parcours arrière (en ayant connaissance des horaires). Pour cette étiquette, le coût correspond à la somme des coûts provenant des algorithmes A_5 et A_4 et l'horaire correspond à l'heure minimale provenant de ces deux algorithmes car les parcours sont effectués en arrière. Lorsque les algorithmes A_2 et A_3 marquent un même nœud, un point potentiel de rencontre est détecté. Son étiquette est insérée pour l'algorithme A_1 qui est également exécuté en parcours arrière. Le cout de cette étiquette correspond à la somme des coûts en provenance des deux algorithmes et l'horaire est obtenu par l'algorithme A_3 (l'algorithme A_2 est exécuté en statique). La méthode s'arrête lorsque l'algorithme A_1 marque l'origine du piéton.
De la même façon que pour la variante d'arrivée au plus tôt du 2SPSPP, il faut utiliser un algorithme multi-label pour calculer le dernier trajet du piéton (ici le trajet calculé par A_1) et résoudre un problème de type *meilleure origine*. Deux règles de dominance pour le calcul de l'heure de départ au plus tard peuvent alors être caractérisées, l'une heuristique, l'autre exacte. Dans le cas du problème 2SPSPP avec départ au plus tard, $LD_{o,d}(t)$, on cherche à maximiser l'heure de départ puisque plus l'heure est grande plus le départ se fera tard, le coût total de la solution sera alors de meilleure qualité. Ainsi, par rapport aux règles utilisées pour l'heure d'arrivée au plus tôt, les conditions sur les temps sont inversées.
Nous définissons alors les règles de dominance de la variante $LD_{o,d}(t)$du 2SPSPP de la même façon que celles utilisées dans la variante $EA_{o,d}(t)$.
Soient deux étiquettes $l_x = (\pi_x, \tau_x)$ et $l'_x = (\pi'_x, \tau'_x)$ associées à un même nœud x, on

dira que l_x domine l'_x, i.e. $l_x \prec l'_x$,

- si $\tau_x \geq \tau'_x \ \wedge \ \pi_x \leq \pi'_x$ pour la dominance heuristique ;
- si $\tau_x \geq \tau'_x \ \wedge \ \pi_x - \pi'_x \leq -(\tau_x - \tau'_x)$ pour la dominance exacte.

La différence entre ces deux règles de dominance est illustrée à la figure 4.6 où l'arc (y, x) est parcouru par l'algorithme du nœud x vers le nœud y. Pour arriver en x à $8h28$, il y a un départ au plus tard de y à $8h25$ correspondant à un temps de trajet de 3. Pour arriver en x à $8h32$, il y a un départ au plus tard de y à $8h25$ correspondant à un temps de trajet de 3 et une attente de 4.

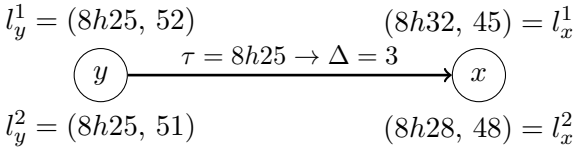


FIGURE 4.6 – Exemple d'application des règles de dominance

En utilisant la règle de dominance heuristique sur le nœud x, on obtient $l_x^1 \prec l_x^2$ car $8h32 \geq 8h28$ et $45 \leq 48$ ce qui conduit à ne pas étendre l'étiquette l_x^2 . La règle de dominance exacte ne peut, quant à elle, pas discriminer ces deux étiquettes : en effet, bien que $8h32 \geq 8h28$, la condition $45 - 48 \leq -(8h32 - 8h28)$, soit $-3 \leq -4$, n'est pas vérifiée, ainsi pour la règle de dominance exacte $l_x^1 \not\prec l_x^2$. De plus, $l_x^2 \not\prec l_x^1$ puisque $8h28 \not\geq 8h32$. Avec la règle de dominance exacte, les deux étiquettes sont donc étendues vers le nœud y pour fournir $l_y^1 = (8h25, 52)$ et $l_y^2 = (8h25, 51)$. Dans ce cas, l'étiquette l_y^2 fournit une solution de meilleur coût que celle fournie par l'étiquette l_y^1 pour un même horaire. La règle de dominance heuristique ne permet pas d'obtenir cette étiquette l_y^2.

La validité de la règle de dominance exacte pour la variante $LD_{o,d}(t)$ se montre de la même façon que pour la variante $EA_{o,d}(t)$.

Soient $l_x = (\pi_x, \tau_x)$ et $l'_x = (\pi'_x, \tau'_x)$ deux labels, associés à un nœud x, tels que $l_x \prec l'_x$. D'après la règle de dominance exacte, on a les inégalités ci-dessous :

$$\tau_x \geq \tau'_x \quad \wedge \quad \pi_x - \pi'_x \leq -(\tau_x - \tau'_x) \quad \Longleftrightarrow \quad \tau_x \geq \tau'_x \quad \wedge \quad \pi'_x - \pi_x \geq \tau_x - \tau'_x \quad (4.1)$$

Soit y un nœud du graphe, et soit $c_y(t)$ le coût du plus court chemin permettant d'arriver au plus tard en x à l'heure t en provenance de y, i.e. $LD_{y,x}(t)$. Puisque le graphe est FIFO, qu'il est parcouru dans le sens inverse pour le calcul $LD_{y,x}(t)$ et que $\tau_x \geq \tau'_x$, on a l'inégalité suivante :

$$\tau_x - c_y(\tau_x) \geq \tau'_x - c_y(\tau'_x) \quad \Longleftrightarrow \quad c_y(\tau'_x) \geq c_y(\tau_x) - (\tau_x - \tau'_x) \quad (4.2)$$

En ajoutant $c_y(\tau'_x)$ dans le deuxième terme de l'équation de dominance (4.1), on

obtient : $c_y(\tau'_x) + \pi'_x \geq \pi_x + c_y(\tau'_x) + (\tau_x - \tau'_x)$.

En combinant cette inégalité avec l'inégalité (4.2) de la condition FIFO, cela donne :

$$c_y(\tau'_x) + \pi'_x \geq \pi_x + c_y(\tau_x) - (\tau_x - \tau'_x) + (\tau_x - \tau'_x) \quad \Longleftrightarrow \quad c_y(\tau'_x) + \pi'_x \geq \pi_x + c_y(\tau_x)$$

Ce résultat traduit le fait que le coût de tous les sommets de l'arbre enraciné en l'_x sont au moins égaux à ceux de l_x, garantissant l'optimalité de la règle de dominance :

$$\forall x, \quad si \quad l_x \prec l'_x \quad alors \quad \forall y, \quad \pi_x + c_y(\tau_x) \leq \pi'_x + c_y(\tau'_x) \quad \wedge \quad \tau_x - c_y(\tau_x) \geq \tau'_x - c_y(\tau'_x)$$

Ce qui peut alors être réécrit en passant aux valeurs en y (puisque $\pi_x + c_y(\tau_y) = \pi_y$ et que $\tau_x - c_y(\tau_x) = \tau_y$ et de même pour les valeurs primées) :

$$\forall x, \quad si \quad l_x \prec l'_x \quad alors \quad \forall y, \quad \pi_y \leq \pi'_y \quad \wedge \quad \tau_y \geq \tau'_y$$

Il est donc toujours inutile d'étendre un label dominé d'après la règle exacte puisque si $l_x \prec l'_x$ alors toute étiquette engendrée par l_x aura un meilleur coût que celle obtenue à partir de l'_x, ce qui est vrai pour le nœud destination en particulier.

De cette façon, il est possible d'adapter le `2SPSPP` au cas où les deux usagers veulent partir le plus tard possible pour arriver chacun à leur destination à leur heure respective voulue.

L'algorithme générique défini dans la section 4.1.3 peut être appliqué pour résoudre la variante $LD_{o,d}(t)$ du `2SPSPP` en utilisant les fonctions définies dans le pseudo-code de l'algorithme 10 à la place de celles utilisées pour la variante $EA_{o,d}(t)$.

La fonction Précédant effectue le parcours arrière de graphe. Les fonctions Domine Heuristiquement et Domine Exactement instaurent les règles de dominance présentées ci-dessus. Enfin, la fonction Rencontre est modifiée afin de prendre le minimum des horaires au lieu de prendre le maximum (lors de la détection d'un point de séparation pour la variante $LD_{o,d}(t)$). Par ailleurs, les fonctions f^{ld} et g^{ld} doivent être passées à l'algorithme générique pour donner les coûts et les horaires dans le passé et non plus dans le futur.

4.2.4.2 `2SPSPP` avec heure de départ et heure d'arrivée

Dans la partie précédente, nous avons vu que les calculs des points de synchronisation pour le `2SPSPP` avec des heures de départ au plus tard ou pour le `2SPSPP` avec des heures d'arrivée au plus tôt sont des problèmes symétriques modulo le changement de la règle de dominance en fonction du problème et du sens de parcours qu'il implique. Nous considérons maintenant la situation combinant une heure de départ pour l'un des usagers, souhaitant arriver au plus tôt et une heure d'arrivée pour l'autre, désirant partir au plus tard. Cette situation peut se produire dans le cas où un conducteur, qui est déjà

Algorithme 10 Fonctions pour l'algorithme 2SPSPP avec heures de départ au plus tard

function PRÉCÉDANT($G = (V, E), x \in V$)
    retourner $\{(y, (y, x)) \mid (y, x) \in E\}$
Fin function

function DOMINE HEURISTIQUEMENT((c^+, τ^+), (c_{ref}, τ_{ref}))
    retourner $c^+ > c_{ref} \wedge \tau^+ > \tau_{ref}$
Fin function

function DOMINE EXACTEMENT((c^+, τ^+), (c_{ref}, τ_{ref}))
    retourner $c^+ > c_{ref} \wedge (c^+ - c_{ref}) < -(\tau^+ - \tau_{ref})$
Fin function

function RENCONTRE(x, $\{H^i\}_{i\in[1,5]}$)
    $(c_1, \tau_1, -, -) \leftarrow H_x^1$
    $(c_2, \tau_2, -, -) \leftarrow H_x^2$
    retourner $(\textbf{Vrai}, c_1 + c_2 + |\tau_1 - \tau_2|, min(\tau_1, \tau_2))$
Fin function

en route, souhaite arriver au plus tôt à sa destination et accepte sur le reste de son trajet un passager qui aimerait arriver à sa destination avant une heure fixée.
Dans ce cadre, nous considérons que le trajet du conducteur ne dépend pas du temps afin que l'approche 2SPSPP soit adaptable. La figure 4.8 détaille le schéma de la situation où le piéton souhaite arriver à l'heure τ_p alors que le conducteur part à l'instant τ_v et il faut minimiser la somme des temps de trajet des deux usagers. Les δ correspondent aux temps de parcours le long des trajets qu'ils étiquettent alors que les h représente les heures de passage aux points indiqué. Les flèches permettent de distinguer les trajets qui sont faits vers l'avant de ceux qui sont faits vers l'arrière.

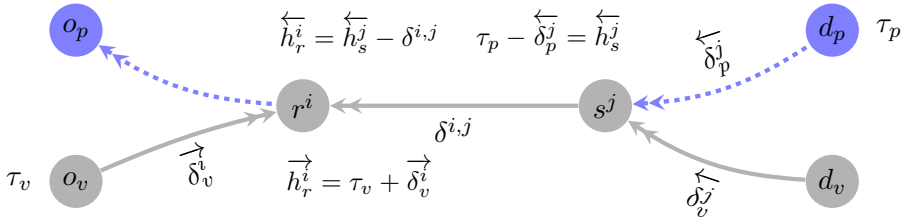


FIGURE 4.7 – Principe de résolution du 2SPSPP pour un départ au plus tard du piéton et une arrivée au plus tôt de la voiture

La méthode proposée consiste à mêler les deux approches décrites précédemment pour les variantes $EA_{o,d}(t)$ et $LD_{o,d}(t)$. Cependant, dans ce cas, il est possible que le problème ne possède pas de solution en raison des horaires fixés par les deux usagers. Il est donc nécessaire d'ajouter des conditions dans cette nouvelle méthode pour vérifier que les solutions potentielles explorées au fur et à mesure sont compatibles en termes d'horaires.
Le principe de résolution est le suivant. Initialement si les horaires d'arrivée au plus tard et de départ au plus tôt sont compatibles, ie. $\tau_v \leq \tau_p$, les algorithmes A_2, A_4, A_5, pour les

trajets respectifs $o_v - r$, $s - d_v$ et $s - d_p$, sont lancés. Autrement, si les horaires initiaux ne sont pas compatibles, il n'est jamais possible que les deux usagers se rencontrent. L'exploration de l'algorithme A_2 se fait en évaluation avant alors que les deux autres (algorithmes A_4 et A_5) se font en parcours arrière, dans un contexte dépendant du temps pour le piéton et en statique pour le conducteur. Pour chaque point de séparation s^j, l'heure de départ $\overleftarrow{h_s^j}$ pour que le piéton arrive à sa destination avant τ_p est alors connue. Il est nécessaire de tester si cet horaire est supérieur ou égal à la date de départ du conducteur, ie. $\tau_v \leq \overleftarrow{h_s^j}$. Si cette condition est vérifiée, alors le point de séparation s^j est inséré dans l'algorithme A_3 pour effectuer une recherche arrière. Pour chaque point de rencontre r^i marqué par les algorithmes A_2 et A_3, deux horaires sont obtenus, le premier $\overrightarrow{h_r^i}$ provient de l'horaire de départ du conducteur et le second $\overleftarrow{h_r^i}$ provient de l'horaire d'arrivée du piéton. Il convient alors de vérifier la compatibilité de ces deux horaires avec la condition $\overrightarrow{h_r^i} \leq \overleftarrow{h_r^i}$. Si elle est vérifiée, l'algorithme A_1 débute une exploration arrière vers l'origine o_p du piéton depuis ce point de rencontre.

Une difficulté spécifique survient à ce stade de la méthode car l'objectif considéré est de minimiser la somme des temps de trajet. Si les deux horaires ne sont pas égaux un temps d'attente $|\overleftarrow{h_r^i} - \overrightarrow{h_r^i}|$ apparaît alors et l'on peut noter qu'il est fixe. En effet, l'attente peut se dérouler sur le point de rencontre, par exemple lorsque le conducteur attend le piéton, ou être décalé sur le trajet entre le point de séparation et la destination du piéton, par exemple lorsque le piéton arrive à l'heure attendue par le conducteur. Le piéton sera alors en avance par rapport aux calculs effectués sur la fin de son trajet vis à vis de sa date d'arrivée au plus tard. Ce temps d'attente peut également se répartir entre les deux usagers. Cependant, puisque l'algorithme A_1 dépend du temps, la répartition de ces temps d'attente entre les usagers va influencer le coût pour la partie du trajet $r^i - o_p$ et donc la fonction objectif. Il convient donc de déterminer quelle politique de répartition du temps d'attente minimise la somme des temps de trajet des deux usagers.

Deux premières stratégies peuvent alors être mises en place pour le choix de l'heure $\overline{h_r^i}$ de l'étiquette insérée dans l'algorithme A_1 suivant qu'il s'agisse :

- de l'heure maximale $\overleftarrow{h_r^i}$, dans ce cas le conducteur attendra au point de rencontre r^i pour une durée de $\overleftarrow{h_r^i} - \overrightarrow{h_r^i}$; en complément de l'objectif global, on cherche alors à minimiser la durée de trajet du piéton ;
- de l'heure minimale $\overrightarrow{h_r^i}$, dans cette situation le piéton devra partir plus tôt que nécessaire de son origine et arrivera à sa destination en avance de $\overleftarrow{h_r^i} - \overrightarrow{h_r^i}$ au maximum selon les horaires des transports en commun le long du trajet $s^j - d_p$; en complément de l'objectif global on vise alors à minimiser la durée de trajet du conducteur.

De façon générale, il est possible d'opter pour une politique de partage du temps d'attente en faisant intervenir un coefficient réel α_i compris entre 0 et 1, pour fixer l'heure au point de rencontre $\overline{h_r^i} = \alpha_i \times \overleftarrow{h_r^i} + (1 - \alpha_i) \times \overrightarrow{h_r^i}$ (on a bien $\overrightarrow{h_r^i} \leq \overline{h_r^i} \leq \overleftarrow{h_r^i}$). Ce coefficient peut dépendre du nœud considéré, et suivant sa valeur, la méthode sera

avantageuse (respectivement désavantageuse) pour le piéton quand il est supérieur (respectivement inférieur) à 0.5 et inversement pour le conducteur.

Si on s'intéresse à un cas spécifique de covoiturage dans lequel on suppose que le conducteur n'attend pas au point de rencontre, il suffit de fixer l'heure $\overline{h_r^i}$ à l'horaire d'arrivée du véhicule $\overrightarrow{h_r^i}$. L'algorithme A_1 peut alors déterminer le coût du trajet entre l'origine du piéton et le point de rencontre sur la base de cet horaire. En revanche, dans le cas général, ces stratégies de répartition des temps d'attente ne garantissent pas d'obtenir la solution optimale, en terme de somme des temps de trajet des deux usagers.

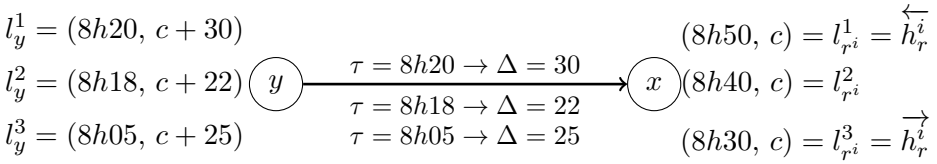


FIGURE 4.8 – Exemple de gain de coût en avancant le départ au plus tard du piéton

Une première approche sous optimale serait d'obtenir des bornes en insérant dans l'algorithme A_1 chaque nœud r^i deux fois. La première avec l'heure $\overrightarrow{h_r^i}$ et la seconde avec l'heure $\overleftarrow{h_r^i}$. La sous-optimalité vient du fait qu'il peut y avoir dans un temps compris entre les deux bornes dans une table horaire du graphe et qui possède un meilleur coût que les deux autres comme cela est illustré sur la figure 4.8. Dans ce cas, l'arc est bien FIFO mais c'est arriver à $8h40$ qui prend le moins de temps de trajet et c'est le choix de cet horaire qui doit motiver la répartition des coûts.
En considérant que plusieurs horaires sont possibles pour un point de rencontre r^i, plusieurs labels sont insérés dans la liste des candidats de l'algorithme A_1. Ces différents labels ont tous le même coût (puisque le temps d'attente est toujours comptabilisé dans le coût quelle que soit la personne qui attend) et des horaires différents. Soient deux labels $l_x = (\pi_x, \tau_x)$ et $l'_x = (\pi'_x, \tau'_x)$ associés à un nœud de rencontre r^i tels que $\pi_x = \pi'_x$. En appliquant la règle de dominance exacte : $\tau_x \geq \tau'_x \quad \wedge \quad \pi_x - \pi'_x \leq -(\tau_x - \tau'_x)$, on obtient $\tau_x \geq \tau'_x \quad \wedge \quad \tau_x \leq \tau'_x$. Ce qui équivaut à $\tau_x = \tau'_x$. Ainsi, la règle de dominance exacte ne permet pas de discriminer des étiquettes ayant des heures de départ différentes et des coûts identiques.

La règle de dominance heuristique ne garderait quant à elle que l'étiquette avec l'horaire le plus elevé puisque $\tau_x \geq \tau'_x \quad \wedge \quad \pi_x \leq \pi'_x$ se ramène à $\tau_x \geq \tau'_x$. Il n'est donc pas possible d'utiliser cette règle qui avorterait d'office toute tentative d'étendre plusieurs horaires et seule la dominance exacte peut être utilisée. Puisque des étiquettes avec des coûts identiques mais des heures différentes ne se dominent pas, si on considère plusieurs horaires possibles pour un point de rencontre, la solution obtenue serait celle dont l'horaire au point de rencontre minimiserait le coût du trajet $r - o_p$.

Toutefois, les fonctions temporelles qui sont utilisées dans les réseaux de transport sont des fonctions en escalier, croissantes si la propriété FIFO est respectée, ce qui est supposé. De cette façon, il va y avoir en chaque nœud de transport en commun un tassement des horaires pour se caler sur les heures de passage des véhicules de transport. Plutôt que de propager plusieurs labels dès le début de la recherche, une possibilité serait de n'effectuer ce dédoublement de labels qu' à partir des nœuds de transport puisque c'est bien lors de la traversée des arcs de transports que des variations de coûts et d'horaires vont avoir lieu.
Une autre piste à considérer consisterait à effectuer une analyse a posteriori considérant le choix de la meilleure origine parmi un ensemble de nœuds pour le trajet du piéton comme un problème à part entière à résoudre sur des entrées fixes et pas en extension par une recherche de type Dijkstra comme c'est le cas actuellement.

Symétries Comme dans les extensions exposées précédemment, il existe une symétrie entre les heures de départ et d'arrivée. Si c'est le conducteur qui doit partir au plus tard pour arriver à sa destination à une heure donnée et le piéton qui part à une heure donnée, il suffit d'appliquer l'algorithme présenté juste avant mais de façon symétrique. Un calcul en parcours avant de l'origine du piéton, avec les horaires va rencontrer l'exploration en parcours avant de la voiture depuis son origine, sans dépendance au temps. Puis ces deux algorithmes vont fusionner pour aller toucher l'exploration en parcours arrière statique de la voiture depuis sa destination avec une heure donnée. Puis quand ces recherches se rencontront elles exploreront le graphe vers la destination du piéton avec une heure de séparation qui dépendra de la politique choisie parmi les différents horaires présentés.
Il est donc bien possible d'établir ce calcul où les heures de départ et d'arrivée des participants ne se trouvent pas du même «côté» (soit à l'origine, soit à la destination).

Une application de cette symétrie est la limite du détour effectué par le conducteur. Dans l'hypothèse où les deux participants ont un horaire donné à leur origines respectives et veulent arriver le plus tôt possible à destination. Il est possible de lancer un calcul de plus court chemin basique uniquement pour la voiture afin de déterminer son heure d'arrivée au plus tôt si elle ne doit pas passer prendre le passager. Si le conducteur ne désire pas effectuer un détour afin de prendre en charge ce piéton, une façon de procéder serait pour lui de chercher à partir le plus tard possible pour arriver à sa destination à l'heure donnée par le calcul de plus court chemin plus cette valeur de détour maximal, ou légèrement avant. Cette convertion permet de se ramener directement au cas où le piéton a une heure de départ et la voiture une heure d'arrivée. Si l'algorithme proposé ne trouve pas de solution, alors cela veut dire que la voiture ne peut prendre en charge le piéton dans la contrainte de temps de détour qu'elle s'est imposée.

4.2.5 Contrainte sur le détour du conducteur

Lors de la résolution du 2SPSPP, il n'y a aucune garantie sur la qualité des trajets individuels de chacun des usagers. Les solutions obtenues, bien qu'optimales en termes d'heures d'arrivées à destination, peuvent ainsi être peu intéressantes pour eux, par exemple si elles entrainent un long détour pour le conducteur ou si elles correspondent à un trajet plus long que le trajet qu'aurait effectué le piéton tout seul s'il n'avait pas pris de covoiturage. Bien que souvent dans la pratique le passager soit avantagé par un transport en covoiturage (par rapport à un trajet en transport en commun), le conducteur est généralement ralenti par la prise en charge d'un passager en dehors de cas particuliers. Nous supposons que le partage de véhicule n'est pas gratifié et qu'il est acceptable pour un automobiliste d'effectuer un détour limité dans le temps par rapport à son trajet le plus court dans le but de faire vivre le partage de son mode de transport. Le nouveau problème considéré consiste donc à trouver des points de rencontre et de séparation optimaux sous la contrainte que le conducteur ne doive pas effectuer un détour supérieur à son chemin le plus court augmenté d'un seuil défini, qu'il s'agisse d'une valeur directe ou d'un pourcentage par rapport à son plus court chemin.

4.2.5.1 Principe

Supposons que le conducteur ne veuille pas faire un détour supérieur à δ et soit c^* le coût de son plus court chemin entre o_v et d_v, alors $c_{max} = \delta + c^*$ est le coût maximal qu'il tolère pour un trajet de covoiturage.

Nous notons également c_x^i le coût au nœud x pour chacun des i algorithmes de la procédure résolvant le 2SPSPP et $C_{x,y}^v$ le coût final de la voiture passant par le point de rencontre x et par le point de séparation y. On a alors $C_{x,y}^v = c_x^2 + c_y^3 + c_y^4$.
L'objectif de l'approche proposée ici est de supprimer, pour chaque sommet x, les étiquettes l_x qui aboutissent à des coût finaux $C_{x,y}^v$ supérieurs à c_{max} dans les algorithmes A_2 et A_3.

Pour effectuer cela, nous allons utiliser une étape de pré-calcul pour déterminer le coût de chaque sommet x jusqu'à la destination du conducteur d_v. Cette première étape consiste à lancer un algorithme de Dijkstra depuis la destination vers l'origine de la voiture en parcours arrière et dont la condition terminale est modifiée. Une fois l'origine trouvée, le coût c^* du plus court chemin est connu mais l'algorithme ne s'arrête pas et il se poursuit tant qu'une étiquette de coût supérieur à c_{max} n'est pas traitée. Ainsi, pour chaque nœud ayant eu une étiquette traitée le coût du plus court chemin jusqu'à la destination c_x^* de la voiture est connu.
Lorsque les algorithmes de Dijkstra A_2 et A_3 s'exécutent, leurs explorations sont limitées par le fait que seuls les nœuds pour lesquels la somme du coût courant et du coût optimal à destination est inférieure à la valeur limite, $c_x^2 + c_x^* \leq c_{max}$ (respectivement $c_x^3 + c_x^* \leq c_{max}$), sont empilés dans le tas. Ainsi, puisque seuls les nœuds respectant

la contrainte sont pris en compte, les points de synchronisation choisis respectent la contrainte de détour.
Le pseudo-code 11 montre le détail de la fonction de filtrage qu'il convient d'utiliser dans l'algorithme générique 8 pour qu'il intègre la contrainte de détour pour le conducteur. Dans le cas où la fonction est appelée sur le deuxième algorithme alors le filtrage n'opère que si la somme du coût courant plus le coût du plus court chemin à la destination est supérieur strictement au coût du chemin le plus court pour la voiture plus son détour maximal autorisé. Si la fonction est appelée sur le troisème algorithme la tâche est légèrement plus complexe car le coût local contient une partie du coût du trajet du piéton depuis l'origine vers le point de rencontre. Il faut donc dans un premier temps récupérer le coût courant de l'algorithme A_3 ainsi que le point de rencontre dont il provient. Puis en utilisant ce nœud, récupérer les coûts du piéton et de la voiture arrivant à lui. Enfin le coût de la voiture s'obtient en soustrayant ces deux coûts à celui de l'algorithme A_3, puis en retranchant un temps d'attente éventuel, et enfin en divisant par deux car le coût est comptabilisé deux fois, une fois pour le piéton et l'autre pour la voiture. Pour finir, le filtrage s'opère si ce coût est supérieur au coût du plus court chemin plus la valeur du détour maximal autorisé.

Algorithme 11 Fonctions pour l'algorithme `2SPSPP` avec contrainte sur le détour de la voiture

function FILTRE DÉTOUR(k, x, $\{H^i\}_{i\in[1,5]}$)
  Si $k = 2$ **alors**
    retourner $\neg(H_x^2[0] + H_x^* \leq c^* + \delta)$
  Fin Si
  Si $k = 3$ **alors**
    $(c, -, r, -) \leftarrow H_x^3$
    $(c_1, \tau_1, -, -) \leftarrow H_r^1$
    $(c_2, \tau_2, -, -) \leftarrow H_r^2$
    $c' \leftarrow (c - c_1 - c_2 - |\tau_1 - \tau_2|)/2$
    retourner $\neg(c' \leq c^* + \delta)$
  Fin Si
  retourner Faux
Fin function

Ainsi, par rapport à l'algorithme générique présenté dans la section 4.1.3, il faut ajouter un algorithme de Dijkstra pour déterminer un isochrone depuis la destination du conducteur et limité par c_{max} et une procédure de filtrage décrite dans l'algorithme 11 pour filtrer les sommets ne respectant pas cette contrainte de détour.

L'algorithme décrit détermine bien les points de rencontre et de séparation de façon optimale car il se base sur l'algorithme `2SPSPP` et seules les étiquettes impliquant un coût final pour la voiture supérieur à celui correspondant à son détour maximal sont conservés. Dans l'algorithme A_3 la priorité est donnée au coûts les plus petits mais comme les nœuds ne sont pas empilés si la contrainte n'est pas respectée ils ne prennent pas la place d'un candidat arrivant avec un coût plus grand mais qui va, lui, bien respecter la contrainte. C'est à dire qu'à partir du moment où un candidat est empilé dans l'un

des algorithmes il existe de façon certaine un chemin allant de ce point à la destination de la voiture respectant la contrainte de détour. Cependant, cela peut être au détriment du passager qui sera potentiellement déposé très loin de sa destination. Charge à lui d'évaluer la pertinence d'un tel covoiturage.
La complexité de cette algorithme est identique à celle de l'approche du `2SPSPP` avec des temps d'exécution potentiellement meilleurs car les espaces de recherche sont réduits par la contrainte.

Une extension serait d'effectuer un filtrage sur l'algorithme A_1 du piéton. En effet, si l'exploration du piéton part dans le sens opposé à celui de la destination de la voiture il serait bien de supprimer ces recherches au bout d'un certain temps. Si les sommets explorés par cette recherche depuis l'origine du piéton ne sont pas marqués par l'algorithme de pré-calcul alors il ne sert à rien de les étendre puisque la voiture ne pourra pas venir récupérer le piéton et arriver à sa destination dans sa limite de détour.
De même un filtrage sur l'algorithme A_4 de la voiture peut être opéré en limitant la taille de l'espace de recherche utilisé dans le pré-calcul. En fait, tout ce qui est fait dans le pré-calcul peut être ré-utilisé dans l'algorithme pour ne pas avoir besoin de lancer la recherche à partir de la destination de la voiture deux fois.

4.3 Analyses expérimentales

Afin de déterminer empiriquement les performances des divers algorithmes présentés et en vue d'obtenir des conclusions générales sur leur utilité, nous avons mené une campagne intensive de tests.

4.3.1 Contexte

Le graphe considéré correspond au réseau de transport de Toulouse et des villes environnantes. Il comporte 76 533 nœuds et 533 152 arcs dont 5 132 de bus, 72 de métro et 77 de tramway. La voirie utilisée est la voirie *OpenStreetMap* et le réseau de transport en commun a été obtenu auprès de Toulouse Métropole, au format *GTFS*.

Les instances de tests sont générées aléatoirement et comportent chacune une origine et une destination pour chacun des usagers. Pour construire une instance deux points de référence r et s sont à déterminer dans le but d'obtenir par la suite les deux points origine o_p et o_v et les deux points destination d_p et d_v qui définissent l'instance. Pour cela, un premier point r est obtenu aléatoirement. Puis un point s est tiré aléatoirement et un plus court chemin en distance entre r et s est calculé. Si ce plus court chemin est compris dans une plage de valeurs données en entrée, alors s est conservé et dans le cas contraire un nouveau point s est choisi aléatoirement jusqu'à ce que la condition soit enfin respectée ou que tous les nœuds du graphe aient été énumérés. Une fois que r et s sont déterminés reste à trouver les quatre points servant à la description de l'instance. Pour les deux points origine, le point o_p (respectivement o_v) est tiré aléatoirement en vérifiant

que sa distance calculée à l'aide d'un algorithme de plus court chemin en distance vers le point r reste comprise dans un intervalle prédéfini donné. De même, pour les points de destination, le point d_p (respectivement d_v) est tiré aléatoirement en vérifiant que sa distance calculée à l'aide d'un algorithme de plus court chemin depuis le point s reste comprise dans un autre plage de valeurs donnée. Ainsi, la donnée des cinq intervalles constitue ce que nous appelerons un scénario et chaque instance est obtenue suivant le principe exposé en se basant sur les données d'un scénario. L'objectif est d'obtenir des groupes d'instances pour lesquels les coûts en distance sur le graphe sont maîtrisés afin de limiter les écarts qui peuvent apparaître d'une instance à l'autre.
Dans tous les scénarios, les distances $o_v - r$, $o_p - r$, $s - d_p$ et $s - d_v$ sont comprises entre 0 km et 5 km. Pour le scénario *1*, la distance $r - s$ est également comprise entre 0 km et 5 km alors que pour le scénario *2* ces valeurs sont fixées à 5 km et 10 km respectivement. Concernant les modes de transport autorisés, le conducteur ne peut utiliser que le mode voiture alors que le piéton est libre d'utiliser tous les transports en commun qu'il souhaite. Puisque le service de transports en commun dépend des heures de la journée, 5 horaires ont été pris en compte pour effectuer les tests ($6:00, 7:00, 8:00, 9:00, 10:00$). Au total 1000 instances ont été générées dans chaque scénario et pour chacun de ces 5 horaires.
Le tableau 4.1 donne un exemple de coût moyen sur les 1000 instances générées du plus court chemin du piéton et de la voiture en l'absence de covoiturage pour chacun des deux scénarios.

Coût du plus court chemin (s)	Scénario 1	Scénario 2
Pour le piéton	2823.0	3789.8
Pour la voiture 2	726.9	761.4

TABLE 4.1 – Coûts moyens des plus courts chemin de l'origine à la destination de chacun des usagers et à 9 : 00

La machine utilisée est une machine 8 octets possédant 32 cœurs et 32G de RAM et exécutant un noyau Linux. Le code a été écrit en *C++* et compilé avec GCC, il fait partie de la plate-forme *PlayMob'*. L'implémentation est mono-processeur mais les différents cœurs peuvent être utilisés pour faire tourner plusieurs instances de l'exécutable.

4.3.2 Sens de parcours

Les premières expérimentations menées consistent à évaluer la performance de la méthode `2SPSPP` dans le cas d'horaires fixés à destination pour le piéton et le conducteur, c'est à dire pour le cas $LD_{o,d}(t)$ du calcul du départ au plus tard. Cette analyse cherche à comparer les règles de dominance exacte et heuristique pour les deux scénarios et les cinq horaires. De plus, ces résultats sont mis en relation avec ceux obtenus pour le cas d'horaires fixés à l'origine, $EA_{o,d}(t)$ du calcul d'arrivée au plus tôt, sur ces mêmes scénarios et horaires.

4.3.2.1 Présentation

Les tableaux 4.4 et 4.5 rassemblent les résultats de nos expérimentations dans le cas $LD_{o,d}(t)$ et pour les scénarios 1 et 2 respectivement. Les tableaux 4.2 et 4.3 fournissent quant à eux les résultats dans le cas $EA_{o,d}(t)$.
La première ligne comprend d'abord les différents horaires ($6:00, 7:00, 8:00, 9:00, 10:00$) dont les colonnes détaillent les valeurs numériques obtenues pour chacun d'entre eux suivant que la règle de dominance utilisée soit exacte «*e*» ou heuristique «*h*». De plus, la colonne «*Moyenne*» donne la moyenne sur les différents horaires de chaque ligne puis la colonne «*Déviation*» fournit l'écart moyen, en pourcentage, entre les solutions obtenues avec la dominance exacte et celles obtenues avec la dominance heuristique.

Les lignes A_1 à A_5 donnent les coûts des solutions (en temps de trajet) pour les différents algorithmes des méthodes. Les lignes «*W*» s'intéressent au temps d'attente. Elles fournissent le nombre de fois où aucun des deux usagers n'attend (ligne «*aucun*»), le nombre de fois où le conducteur attend et son temps d'attente moyen, puis le nombre de fois où le piéton attend et son temps d'attente moyen. La ligne «*coût total*» fournit le coût total du trajet de covoiturage correspondant à la somme des coûts en comptant deux fois le coût de l'algorithme A_3. Les valeurs contenant plus de décimales qu'affichées, les valeurs de la somme peuvent donc être différentes de celles qui pourraient être recalculées à partir du tableau. Toutes les valeurs données précédemment sont en secondes. Enfin, la ligne «*Temps de calcul*» présente le temps de calcul CPU total de l'exécution, en millisecondes. Rappelons que chaque case hormis celle des colonnes «*Moyenne*» et «*Déviation*» représente 1000 instances, toutes différentes mais respectant les contraintes de leur scénario.

4.3.2.2 Analyse des résultats

On peut noter que les coûts des solutions varient en fonction des horaires considérés, ce qui est certainement dû à l'utilisation de modes de transport en commun. L'utilisation de la règle heuristique a un faible impact sur la qualité des solutions que ce soit dans le cas $LD_{o,d}(t)$ ou $EA_{o,d}(t)$ et quel que soit le scénario : la perte de qualité de solution varie de 0.23% à 0.78%. En revanche, le temps de calcul est fortement diminué par l'utilisation de cette règle heuristique : l'économie de temps de calcul allant de 151% à 351%, ce qui correspond à environ 300 ms pour le scénario 1 et 500 ms pour le scénario 2.
Concernant l'attente, les cas où les participants se retrouvent au point de rencontre au même moment sont rares et stable aux alentours de 2%. Sinon, c'est le piéton qui attend majoritairement à hauteur de 30% du temps environ, qu'importe le scénario et le sens de parcours.
Dans le cas $LD_{o,d}(t)$, l'utilisation de la dominance heuristique conduit à une forte diminution du coût du trajet du piéton calculé en parcours arrière du point de rencontre vers

son origine. Elle entraine également une augmentation du coût du trajet du conducteur calculé en parcours arrière du point de rencontre vers son origine et du coût du trajet commun. Les trajets calculés depuis les destinations vers le point de séparation sont quant à eux très faiblement touchés par la règle de dominance utilisée. Le phénomène est totalement symétrique au cas $EA_{o,d}(t)$ où cette fois c'est le trajet du piéton calculé en forward du point de séparation vers sa destination qui diminue fortement avec l'utilisation de la dominance heuristique. En termes d'attente, la dominance heuristique conduit à favoriser le piéton en entrainant dans la majorité des cas une légère baisse de son temps d'attente.

Enfin, et dans tous les cas de figures, la règle de dominance heuristique est bien meilleure que la règle exacte en terme de coût pour l'algorithme A_1 dans le cas $LD_{o,d}(t)$ et pour l'algorithme A_5 dans le cas $EA_{o,d}(t)$. Cela est conforme avec le fait qu'elle ne s'applique que sur le dernier algorithme.

Horaire		6 : 00		7 : 00		8 : 00		9 : 00		10 : 00		Moyenne		Déviation
Dominance		e	h	e	h	e	h	e	h	e	h	e	h	
A_1		521.738	524.081	518.612	519.451	518.398	518.708	516.409	517.037	520.924	521.198	519.216	520.095	0.17%
A_2		551.497	552.345	543.909	543.949	543.724	543.544	544.958	544.953	547.130	546.829	546.243	546.324	0.01%
A_3		499.009	526.965	481.171	503.602	482.245	507.038	483.775	508.381	490.598	514.981	487.359	512.193	4.85%
A_4		561.993	585.784	557.760	577.997	561.334	580.703	558.958	580.018	565.876	583.921	561.184	581.684	3.52%
A_5		193.775	125.488	200.470	147.929	200.984	142.441	207.211	147.466	191.718	135.646	198.831	139.794	-42.23%
Attente	aucun	26	26	26	26	27	28	32	32	27	26	27.6	27.6	
	voiture	271	274	292	296	292	295	290	293	286	289	286.2	289.4	
		70.077	69.094	71.739	72.408	65.976	67.122	56.258	56.273	65.209	64.529	65.852	65.885	0.05%
	piéton	703	700	682	678	681	677	678	675	687	685	686.2	683	
		69.345	67.422	67.807	67.744	65.478	65.933	66.171	65.783	65.292	64.642	66.819	66.305	-0.78%
Coût total		2894.762	2908.318	2850.286	2864.381	2852.786	2863.91	2856.265	2867.128	2870.350	2880.485	2864.890	2876.844	0.42%
Temps de calcul		1191.308	413.111	642.428	306.332	651.002	258.942	461.471	200.339	1168.377	458.785	822.917	327.501	-151.27%

TABLE 4.2 – Résultats pour des arrivées au plus tôt ($EA_{o,d}(t)$), scénario 1

Horaire		6 : 00		7 : 00		8 : 00		9 : 00		10 : 00		Moyenne		Déviation
Dominance		e	h	e	h	e	h	e	h	e	h	e	h	
A_1		540.455	544.045	538.047	542.290	537.174	540.974	539.492	542.486	542.162	543.338	539.466	542.626	0.58%
A_2		573.410	576.022	569.009	571.299	568.122	570.580	568.413	569.378	571.579	573.230	570.106	572.101	0.35%
A_3		661.420	734.809	629.630	693.623	642.739	700.828	645.809	711.709	670.051	720.103	649.929	712.214	8.75%
A_4		775.480	843.397	760.749	819.144	769.231	824.060	772.252	832.922	796.913	842.071	774.925	832.318	6.90%
A_5		511.312	327.277	553.525	391.743	530.036	379.163	528.081	363.130	463.495	337.061	517.289	359.674	-43.82%
Attente	aucun	24	26	19	20	21	24	23	23	25	25	22.4	23.6	
	voiture	284	282	298	296	281	289	304	306	283	283	290	291.2	
		47.968	50.269	50.154	52.547	50.779	52.615	55.851	56.205	55.879	55.455	52.126	53.418	2.42%
	piéton	692	692	683	684	698	687	673	661	692	692	687.6	685.2	
		67.309	66.695	67.215	65.150	64.780	65.228	68.202	65.709	65.362	65.875	66.574	65.731	-1.28%
Coût total		3783.698	3820.749	3741.444	3771.974	3749.527	3776.805	3762.735	3792.954	3775.296	3797.233	3762.540	3791.943	0.78%
Temps de calcul		3253.499	684.300	1768.606	386.157	1769.116	393.664	1182.952	311.269	3292.515	702.511	2253.338	495.580	-354.60%

TABLE 4.3 – Résultats pour des arrivées au plus tôt ($EA_{o,d}(t)$), scénario 2

Horaire		6 : 00		7 : 00		8 : 00		9 : 00		10 : 00		Moyenne		Déviation
Dominance		e	h	e	h	e	h	e	h	e	h	e	h	
A_1		76.887	63.757	171.974	126.160	224.808	160.691	144.592	144.449	217.610	146.378	167.212	128.313	-30.32%
A_2		598.172	601.848	557.077	575.695	546.080	568.617	571.845	571.689	550.088	574.306	564.805	578.588	2.38%
A_3		557.811	563.786	497.724	516.257	475.241	502.308	507.801	507.950	479.814	509.205	503.573	519.801	3.12%
A_4		562.961	563.100	551.089	551.710	551.420	551.185	555.700	555.692	551.591	551.661	554.648	554.765	0.02%
A_5		524.028	523.993	522.568	523.263	521.278	521.714	525.056	525.081	523.609	524.061	523.401	523.716	0.06%
Attente	aucun	20	20	20	20	21	21	16	16	23	22	20	19.8	
	voiture	238	236	297	297	282	283	277	278	294	294	277.6	277.6	
		38.941	39.584	61.579	62.033	55.535	58.180	54.126	53.942	61.010	63.986	54.238	55.545	2.35%
	piéton	738	740	681	681	697	696	707	706	683	684	700.8	701	
		65.102	65.259	68.653	68.743	65.714	66	64.550	64.559	67.231	67.853	66.271	66.512	0.36%
Coût total		2935.216	2938.137	2863.330	2874.835	2855.532	2869.224	2873.545	2873.414	2866.382	2880.062	2878.985	2887.321	0.29%
Temps de calcul		1105.074	265.240	761.957	258.180	761.557	268.17	916.68	220.088	1365.597	484.164	982.357	299.163	-228.37%

TABLE 4.4 – Résultats pour des départs au plus tard ($LD_{o,d}(t)$), scénario 1

Horaire		6 : 00		7 : 00		8 : 00		9 : 00		10 : 00		Moyenne		Déviation
Dominance		e	h	e	h	e	h	e	h	e	h	e	h	
A_1		59.374	45.228	177.156	116.451	230.954	142.386	138.784	140.973	209.300	141.537	163.119	117.318	-39.10%
A_2		627.040	630.805	589.314	606.930	568.434	597.95	599.785	599.572	580.218	599.521	593.026	607.025	2.31%
A_3		810.792	816.851	718.035	744.097	686.675	725.888	735.743	734.717	707.548	737.795	731.789	751.901	2.67%
A_4		834.717	834.337	806.595	806.927	799.066	799.035	805.213	805.199	810.516	810.195	811.167	811.213	0.01%
A_5		789.570	789.847	789.759	790.137	790.093	790.239	791.286	791.340	791.285	792.431	790.402	790.803	0.05%
Attente	aucun	15	15	19	19	18	18	21	21	20	20	18.6	18.6	
	voiture	270	273	384	382	391	392	370	370	357	359	354.8	354.8	
		51.148	52.421	73.773	74.973	91.708	90.096	77.162	77.162	81.376	83.588	75.033	75.612	0.77%
	piéton	713	710	595	597	591	590	609	609	623	621	625.6	625.6	
		82.562	82.692	75.850	75.926	75.856	74.769	69.748	69.637	77.494	76.927	76.321	76.009	-0.41%
Coût total		4005.110	4007.092	3872.503	3882.689	3842.586	3860.818	3877.581	3877.581	3883.432	3897.362	3896.403	3905.249	0.23%
Temps de calcul		2628.433	547.882	1267.939	466.190	1195.702	372.678	1482.102	298.972	2145.811	684.529	1744.240	474.040	-267.95%

TABLE 4.5 – Résultats pour des départs au plus tard ($LD_{o,d}(t)$), scénario 2

4.3.3 Contrainte de détour

L'algorithme proposé en section 4.2.5 permet à l'automobiliste de limiter le détour qu'il fera par rapport à son plus court chemin pour prendre en charge le passager. Afin de valider la méthode empiriquement, c'est à dire vérifier que le détour effectif est toujours inférieur à la contrainte donnée en consigne, nous avons effectué une analyse expérimentale faisant varier plusieurs paramètre. L'objectif est aussi de déterminer comment la contrainte de détour affecte la qualité des solutions obtenues.

4.3.3.1 Contexte expérimental

Dans cette section, les instances considérées sont basées sur les mêmes scénarios et horaires que dans la section précédente (pour chaque scénario et horaire, 1000 instances sont générées aléatoirement). De plus, nous considérons différentes valeurs de détour pour le conducteur allant de 1 à 50 minutes (3000 secondes) auxquelles nous ajoutons le cas du détour infini, c'est à dire l'absence de contrainte de détour pour le conducteur. En réalité, les expérimentations ont porté sur un détour allant jusqu'à des valeurs plus grandes mais les courbes présentées convergent légèrement avant 3000 secondes de détour pour certaines.
Nous nous plaçons dans le cas d'un horaire de départ aux origines des deux usagers dans le but de calculer leur heure d'arrivée au plus tôt à destination. Dans la continuité des expérimentations, les résultats présentés ci-dessous rassemblent 1000 instances par détour, par heure, par dominance et par scénario.
Deux types de graphiques sont présentés. Les premiers détaillent, pour chaque scénario séparément, l'évolution de métriques, présentées en ordonnée, en fonction du détour, fourni en abscisse, pour les différents horaires étudiés en se plaçant uniquement dans le contexte de l'utilisation de la dominance exacte. Le but est alors d'étudier l'impact des horaires sur ces différentes métriques. Les seconds détaillent l'évolution de ces mêmes métriques, toujours en ordonnée, en fonction du détour, mais pour l'horaire unique de 8 : 00 et en distinguant les règles de dominance utilisées et les scénarios.

4.3.3.2 Evaluation du nombre de solutions en fonction du détour

Les graphiques 4.9 (pour le scénario 1) et 4.10 (pour le scénario 2) donnent le nombre de solutions obtenues. Comme attendu, plus la valeur du détour maximal est élevée, plus le nombre de solutions de covoiturage obtenues est important. En effet, lorsque le détour du conducteur est faible, il est possible que les deux usagers ne puissent pas se rejoindre pour effectuer un trajet en covoiturage. Par exemple, pour un départ à 8 : 00, dans le scénario 1, l'ensemble des instances a une solution à partir d'un détour de 3000 secondes soit 50 minutes. Pour le scénario 2, il faut un détour maximal de 2880 secondes pour que toutes les instances aient une solution. Sur ces deux scénarios, les horaires n'ont pas de réelle influence sur le nombre de solutions obtenues.

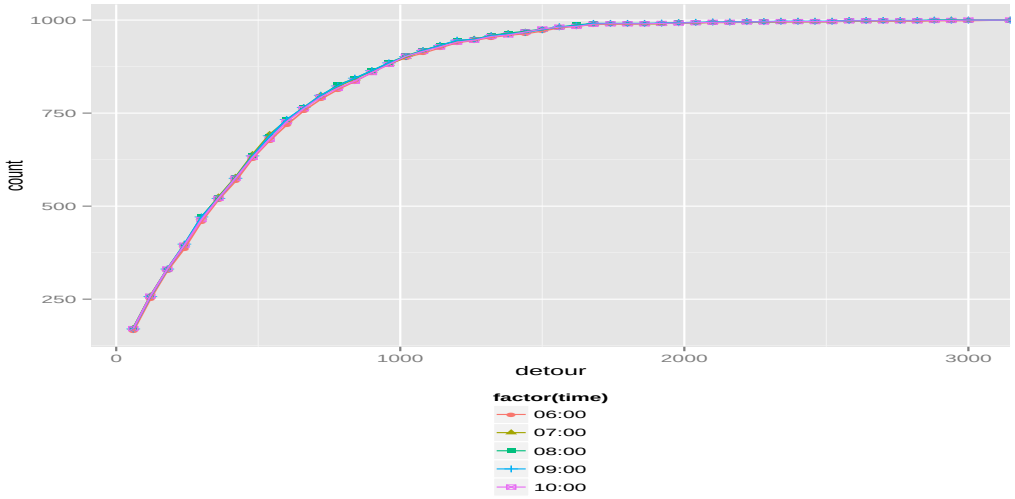


FIGURE 4.9 – Nombre de solutions par horaire de départ pour le scénario 1

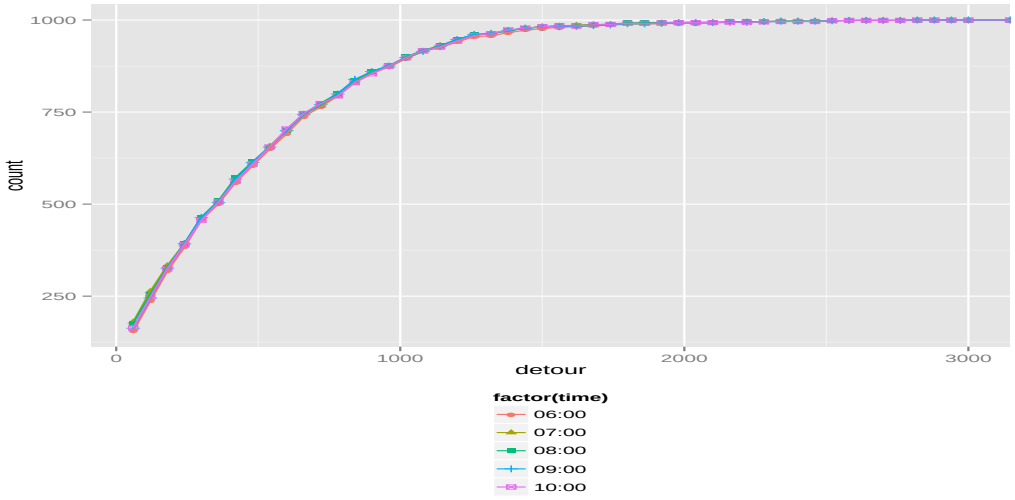


FIGURE 4.10 – Nombre de solutions par horaire de départ pour le scénario 2

Dans le graphique 4.11, nous comparons le nombre de solutions obtenues pour chacun des scénarios, en moyenne pour les cinq horaires, en fonction de l'utilisation des règles de dominance exacte ou heuristique.

Sur ce graphique, nous pouvons noter que le nombre de solutions obtenues ne dépend pas de la règle de dominance utilisée. Pour le scénario 1, en raison des trajets plus courts, il y a un peu plus de solutions pour des valeurs de détour faibles (entre 500 et 1000 secondes) comparativement au scénario 2. Mais globalement il n'y a pas de différence significative entre nos deux scénarios vis-à-vis du détour maximal.

Enfin, le pourcentage de solutions obtenues avec un détour maximal de 1 minute est d'environ 20% ce qui est déjà assez important et ce pourcentage monte à 65% pour un détour 10 fois plus important.

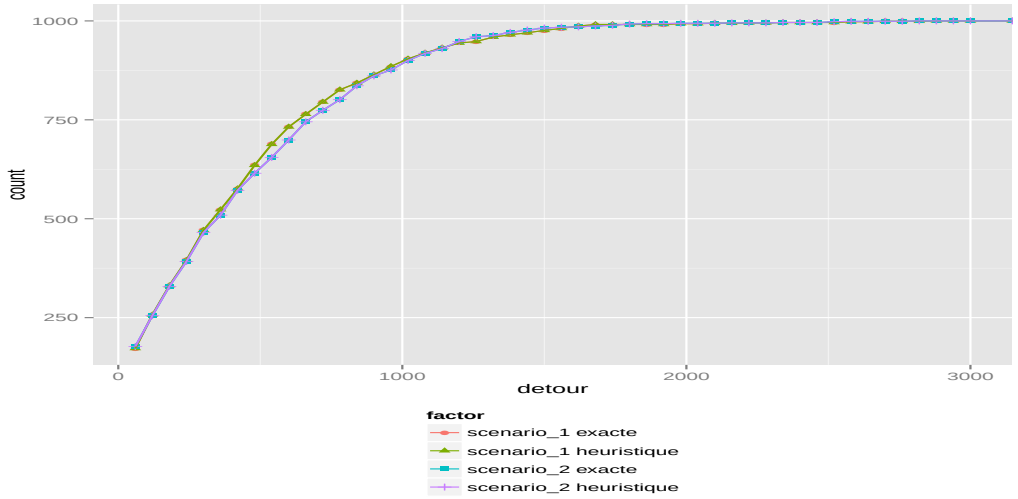


FIGURE 4.11 – Nombre de solutions en fonction des règles de dominance

4.3.3.3 Evaluation du coût des solutions en fonction du détour

Les graphiques 4.12 et 4.13 présentent l'évolution du coût total des solutions de covoiturage en fonction de la valeur limite du détour que se fixe le conducteur pour les scénarios 1 et 2 respectivement. Ils distinguent pour chacun d'entre eux les cinq horaires étudiés.

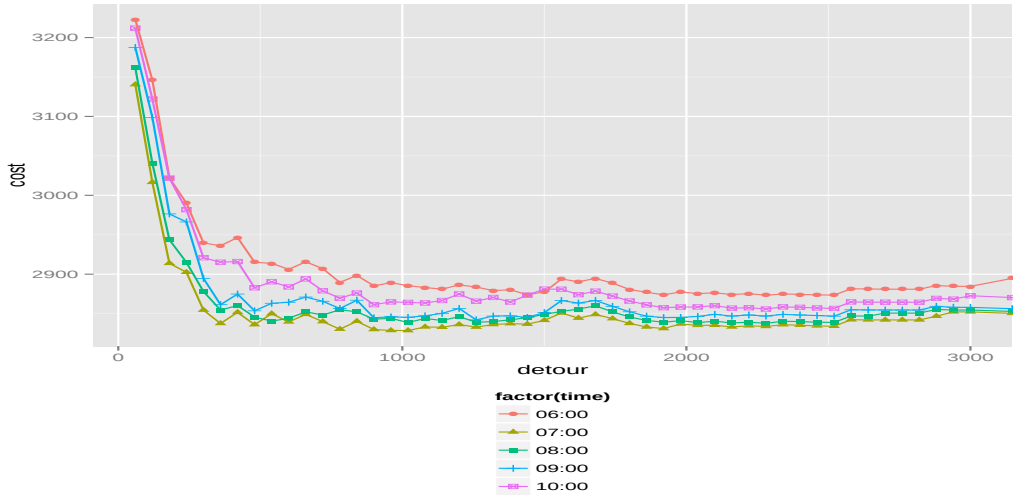


FIGURE 4.12 – Coût des solutions par horaire de départ pour le scénario 1

Globalement, pour ces deux scénarios, plus la valeur maximale de détour augmente et plus le coût des solutions de covoiturage diminue. Ceci correspond au fait que le piéton a un trajet moins important à faire seul. Cette décroissance n'est pas stricte car le coût moyen n'est calculé que pour les instances qui ont une solution pour une valeur donnée de détour. Nous avons cependant vérifié par ailleurs que pour chaque instance et à partir

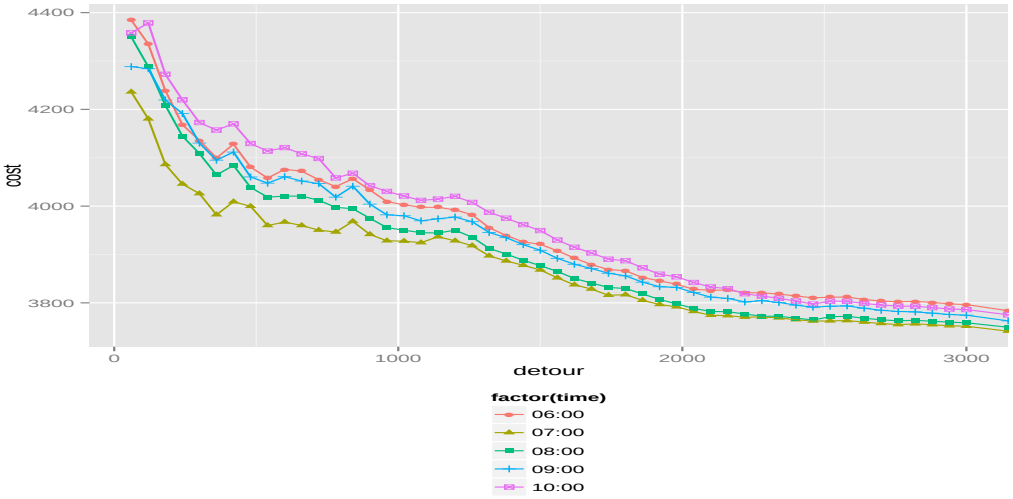


FIGURE 4.13 – Coût des solutions par horaire de départ pour le scénario 2

d'un détour donnant une solution, l'acceptation d'un détour plus important implique la baisse du coût. Le coût des solutions varie en fonction des horaires, les horaires extrèmes (6 : 00 et 10 : 00) conduisent à des solutions de coût plus élevé, ce qui traduit une offre moindre de transport en commun. Dans le cas du scénario 1, le coût converge très rapidement avec le détour effectué par le conducteur et l'on peut dire que les coûts se stabilisent dès 1000 secondes de détour. En revanche, le scénario 2 est plus sensible à la valeur maximale de détour. Le coût des solutions converge beaucoup plus lentement et ne se stabilise qu'aux alentours de 2500 secondes de détour, la distance à parcourir étant plus importante en moyenne.

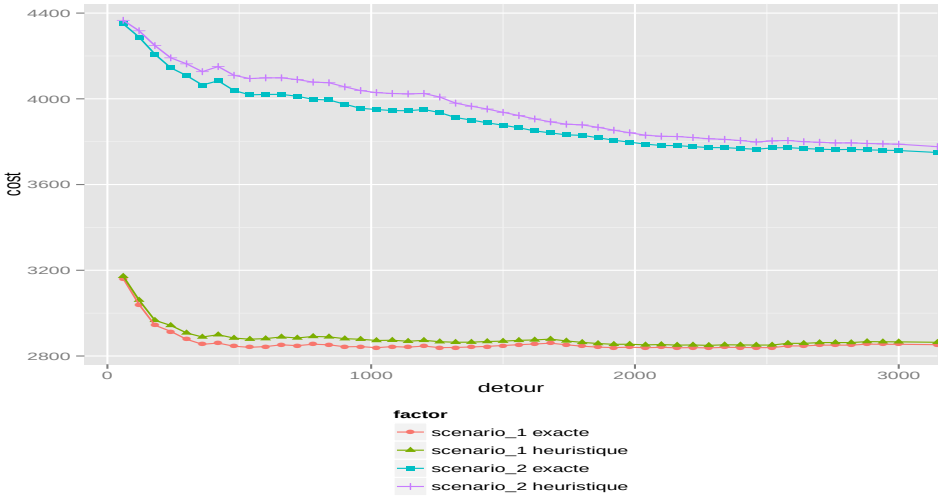


FIGURE 4.14 – Coût des solutions en fonction des règles de dominance

Le graphique 4.14 montre une très modeste différence en terme de coût des solutions

entre la règle de dominance heuristique et la règle exacte. Cela rejoint les observations présentées dans la section 4.3.2. En raison des paramètres de génération des scénarios, le coût des solutions de covoiturage du scénario 2 est plus important que le coût des solutions du scénario 1.

4.3.3.4 Evaluation du détour réel en fonction du détour maximal autorisé

L'un des objectifs des expérimentations vise à vérifier, de manière empirique, que le détour réel effectué par le conducteur, est bien toujours inférieur au détour maximal autorisé. Pour cela, les figures 4.15 et 4.16 décrivent pour chacun des scénarios l'évolution de ce détour réel par rapport à la consigne. La droite identité est également représentée sur ce graphique. Il est donc possible de vérifier que le détour réel moyen reste bien en deçà du détour maximal fixé par le conducteur.

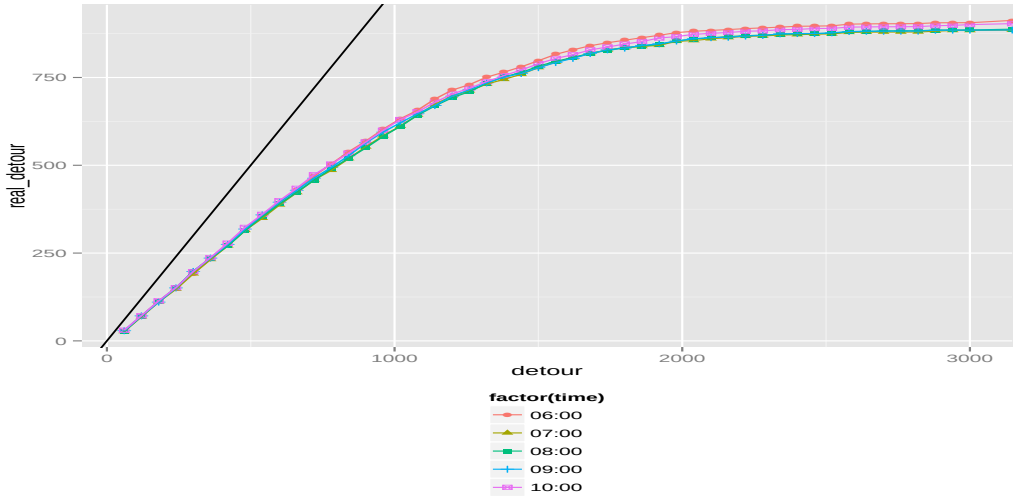


FIGURE 4.15 – Détour réalisé par horaire de départ pour le scénario 1

On peut de plus observer que les horaires n'ont qu'une très petite influence sur le détour réel. Par ailleurs, la valeur du détour réel converge plus lentement pour le scénario 2 (à partir d'un détour maximal de 3000 secondes) que pour le scénario 1 (qui converge dès 2000 secondes de détour maximal). De plus, l'augmentation du détour réel est de type logarithmique.

Enfin, la figure 4.17 montre que la règle de dominance ne modifie pas la forme de la courbe d'évolution du détour réel mais uniquement la vitesse à laquelle elle converge vers la solution pour un détour infini.

4.3.3.5 Evaluation du bénéfice du covoiturage

Pour terminer l'analyse des résultats, nous cherchons à déterminer si le covoiturage est intéressant dans le cadre présenté, c'est à dire sans rémunération pour le conducteur et avec une pondération identique pour le conducteur et le passager. Pour cela, le

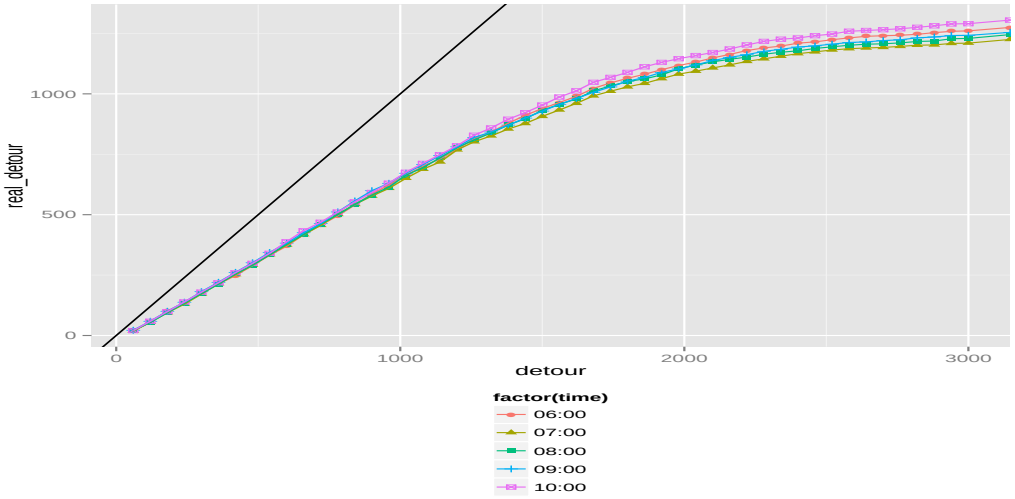


FIGURE 4.16 – Détour réalisé par horaire de départ pour le scénario 2

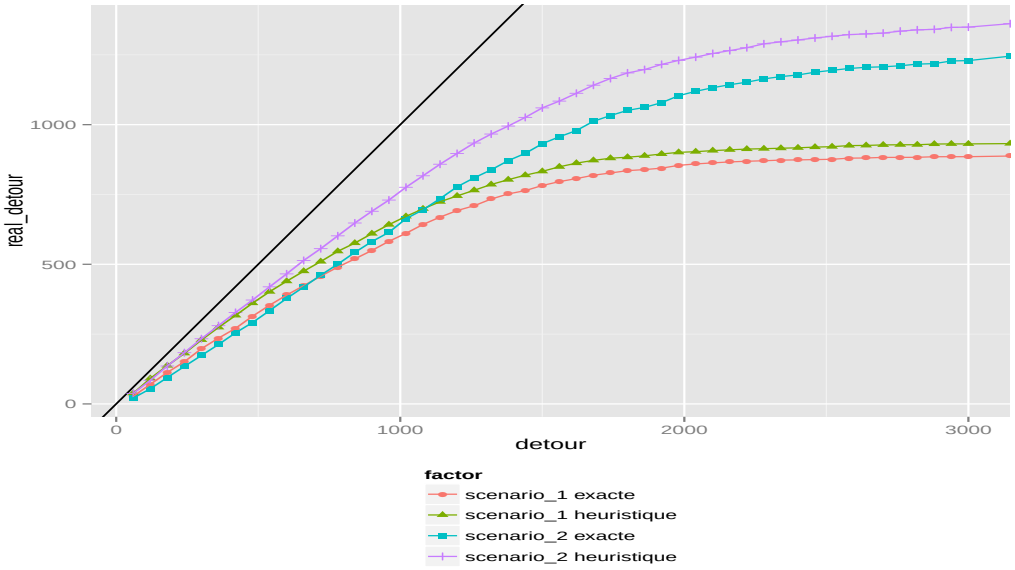


FIGURE 4.17 – Détour réalisé en fonction des règles de dominance

bénéfice lié au covoiturage pour chacun des participants est calculé par rapport à son plus court chemin individuel puis normalisé afin d'obtenir un pourcentage. Puisque le covoiturage permet dans la plupart des cas au piéton d'arriver plus vite à destination grâce à l'utilisation de la voiture par le piéton, celui-ci est en moyenne gagnant alors que le conducteur est généralement ralenti par la prise en charge d'un covoitureur. Le bénéfice global du système est obtenu par la somme des différences des coûts de chacun des participants à leur plus court chemin le tout divisé par le coût total du covoiturage.

Les courbes 4.18 et 4.19 montrent les bénéfices pour les deux scénarios. Les courbes de la partie haute des graphiques correspondent au bénéfice du piéton qui est toujours

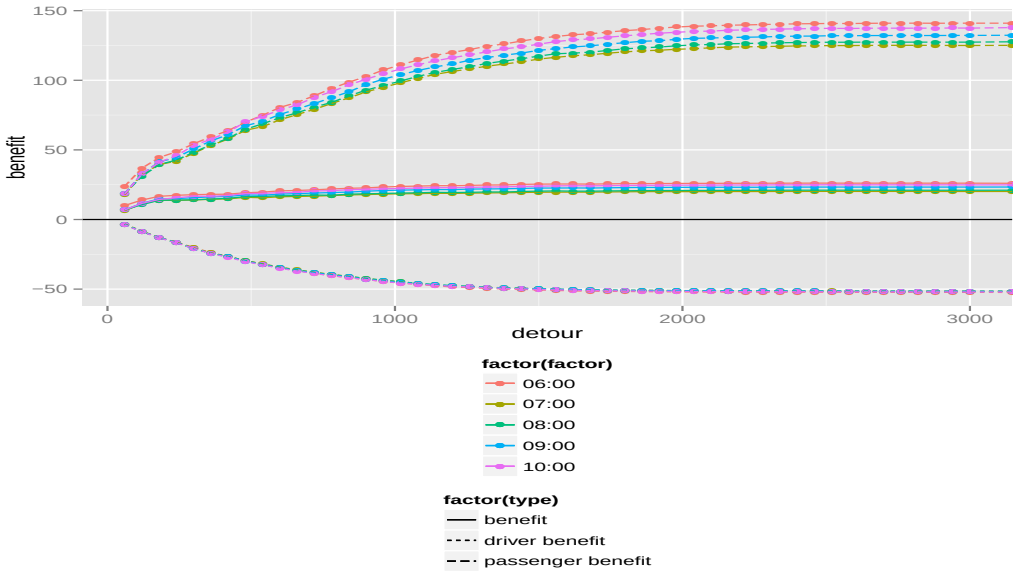


FIGURE 4.18 – Bénéfices par horaires de départ pour le scénario 1

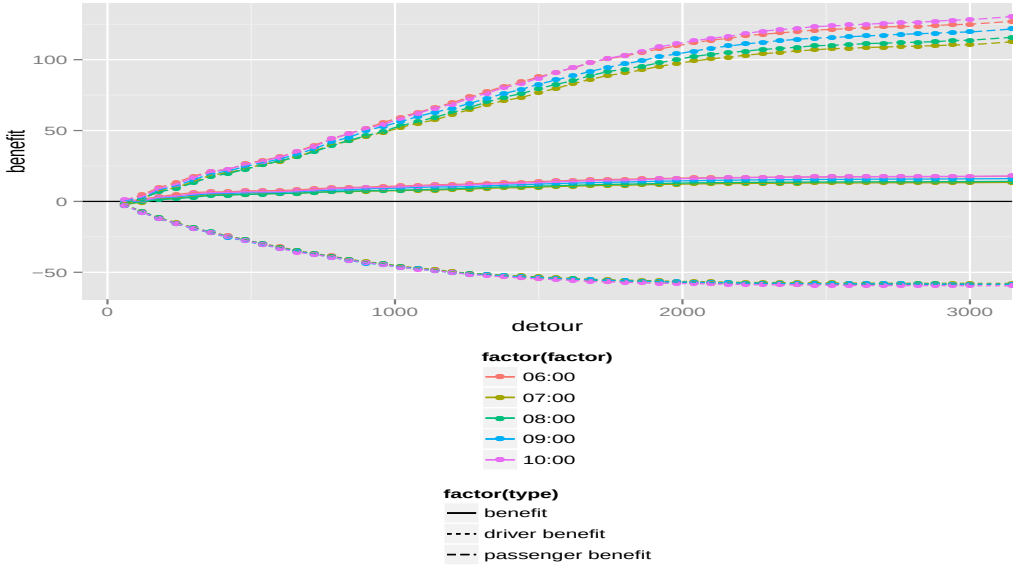


FIGURE 4.19 – Bénéfices par horaires d'arrivée pour le scénario 2

positif et celles de la partie basse représentent le bénéfice du conducteur qui est toujours négatif. Les courbes de la partie centrale traduisent le bénéfice global du système, soit la somme des deux bénéfices précédants. On peut noter que les horaires de départ ont une légère influence sur les bénéfices du piéton, mais qu'ils n'ont aucune influence sur ceux du conducteur dont les trajets sont indépendants des horaires. Sur le bénéfice global du système, les horaires n'ont qu'une faible influence. De plus, pour le scénario 1, la valeur maximale de détour, même si elle influence les bénéfices (en positif ou négatif) de chacun

des usagers, n'a pas d'influence sur le bénéfice global du système. Pour le scénario 2, le bénéfice global croit progressivement en fonction de la valeur maximale de détour.
D'ailleurs, le bénéfice total n'est pas toujours positif concernant le scénario 2. Il est donc possible que le système global soit perdant pour de faibles valeurs de limite de détour.

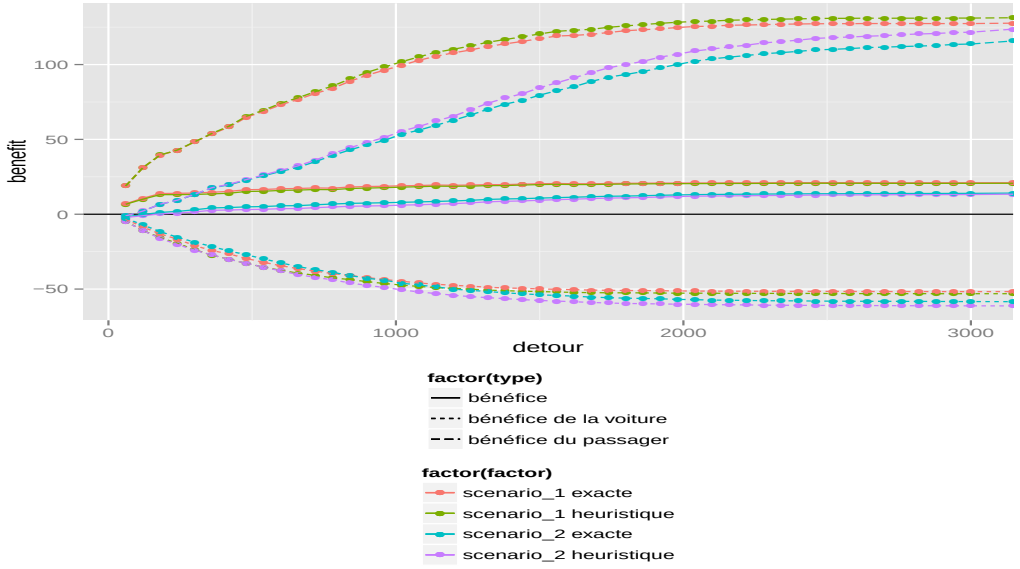


FIGURE 4.20 – Bénéfices en fonction des règles de dominance

La courbe 4.20 indique clairement que la règle de dominance n'influence pas le bénéfice global. En revanche, le type de scénario influence fortement le bénéfice du piéton et très faiblement celui du conducteur.
Pour un détour de 60 secondes, le minimum dans les expérimentations menées, le système est gagnant pour le scénario 1 mais pas pour le scénario 2. Cela est dû au fait que plus un trajet partagé va être important par rapport aux trajets individuels et plus l'impact du covoiturage sera faible. Toutefois, le gain espéré par la pratique du covoiturage sature très rapidement. A partir de 10 minutes de détour, presque 100% du gain espéré est atteint pour le scénario 1 contre 50% pour le scénario 2. Il n'est donc pas rentable d'effectuer des détours trop longs pour la voiture même si elle n'y perd plus rien car le piéton n'y gagne plus non plus.

4.3.3.6 Evaluation du temps de calcul en fonction du détour

Enfin, l'analyse des temps de calcul présentés en figures 4.21 et 4.22 montre qu'il est fortement dépendant de l'horaire considéré. En effet, pour les horaire extrêmes de l'étude, à savoir 6 : 00 et 10 : 00, le temps de calcul est presque deux fois plus important. De plus, le temps de calcul peut varier très fortement, tout comme le coût peut le faire également, en fonction du scénario : plus les trajets sont longs et plus le temps de

résolution est important.

FIGURE 4.21 – Temps de calcul par horaire de départ pour le scénario 1

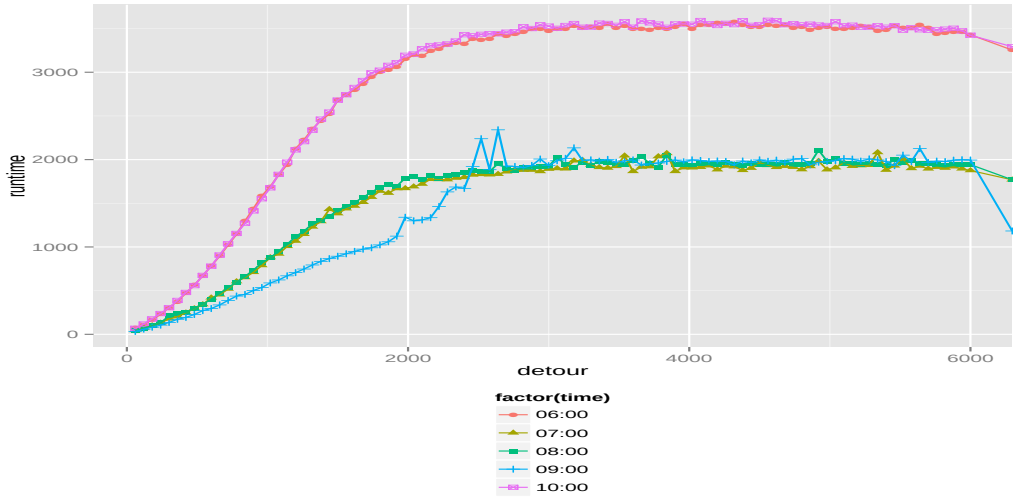


FIGURE 4.22 – Temps de calcul par horaire de départ pour le scénario 2

La courbe 4.23 montre que les temps de calcul entre la dominance exacte et la dominance heuristique diffèrent de façon moins importante pour le scénario 1 que pour le scénario 2. Pour le scénario 1, le temps de calcul peut être jusqu'à 2.5 fois plus important avec la dominance exacte par rapport à la dominance heuristique. Alors que pour le scénario 2, cet écart peut aller jusqu'à 1.5. Pour le scénario 2, la pente de la courbe du temps d'exécution avec la dominance exacte est importante mais ne révèle pas une explosion combinatoire, la forme de la courbe restant identique à celle de la dominance heuristique.

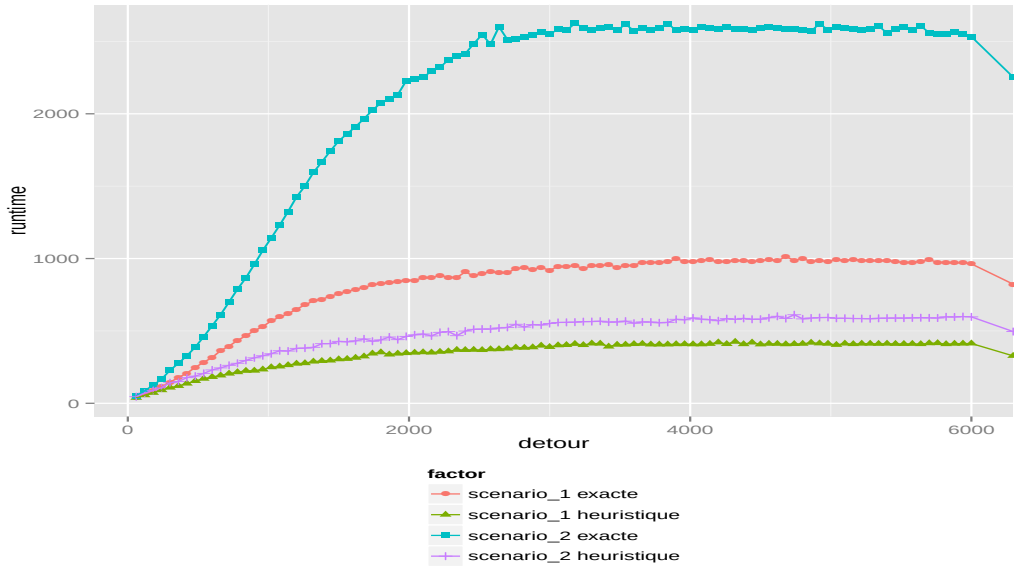


FIGURE 4.23 – Temps de calcul en fonction des règles de dominance

4.3.3.7 Bilan des expérimentations

Cette étude expérimentale concerne des situations de covoiturage en zones urbaines et périurbaines, là où la distance de covoiturage va être plutôt comparable à celle du scénario 1 et très souvent inférieure à celle du scénario 2. Il s'agit donc d'un cas d'étude pour le covoiturage de proximité.
Pour le scénario 2, le nombre de solutions de covoiturage augmente assez rapidement avec la valeur maximale de détour et est assez élevé dès 1 minute de détour. A 10 minutes de détour maximal, le détour réel supporté par le conducteur est de 7 minutes alors que le nombre de solutions est de 65% et le bénéfice global de 5% à 10%. Ces résultats peuvent être obtenus dans des temps de calcul en dessous de la seconde en moyenne sur les différents horaires et scénarios.

4.4 Conclusion

Dans ce chapitre, nous avons présenté des extensions du problème `2SPSPP` en faisant varier les endroits où sont définis les horaires pour les problèmes $EA_{o,d}(t)$ et $LD_{o,d}(t)$. En se basant sur la méthode de résolution proposée dans [26], nous avons parfois montré l'impossibilité d'utiliser ce modèle dans certains cas alors que dans d'autres, la méthode s'adapte très bien. Ainsi, il est possible de résoudre, comme nous l'avons montré avec les règles de dominances ainsi que les évaluations expérimentales, de façon parfaitement symétrique le problème de synchronisation lorsque les deux participants donnent une heure d'arrivée et cherchent à partir au plus tard de leur origine. Deux règles de dominances sont applicables, l'une heuristique, l'autre exacte et les caractéristiques des temps

de résolution, la répartition des coûts parmi les cinq algorithmes ainsi que la probabilité que le piéton ou la voiture attendent sont identiques dans l'approche en parcours avant tout comme dans l'approche en parcours arrière.
Une extension un peu particulière consistant à donner une heure de départ pour l'un et une heure d'arrivée pour l'autre a été étudiée. La propagation du temps d'attente s'établie sur le dernier algorithme et pose de nouvelles problèmatiques qui n'ont été abordées que brièvement. En particulier, le problème de meilleure origine ne se limite plus à partir d'un ensemble de points à heure fixe mais à en partir à de multiples horaires compris des plages de valeurs.
Enfin, la contribution la plus conséquence a porté sur l'intégration d'une limite de détour pour la voiture à l'algorithme de la littérature. Pour cela, un pré-calcul obligatoire permet de respecter cette contrainte à chaque étape de l'algorithme. La solution trouvée, si elle existe, est alors la meilleure garantissant à l'automobiliste qu'il n'effectuera pas un détour supérieur à sa volonté. La campagne expérimentale menée sur ce problème a permis de montrer qu'il existe un seuil sur la valeur maximale de détour à partir duquel il devient bénéfique pour le système que le conducteur effectue un détour modéré par rapport à sa valeur limite sachant que le nombre d'instances pour lesquelles une solution est trouvée devient rapidement assez important en fonction du détour maximal. En deçà, la rentabilité de la méthode n'est pas assurée ; au delà, un gain constant est garanti. Dans le premier scénario de test considéré, le seuil sur la valeur maximale de détour était même égale à 0, le système étant alors toujours bénéficiaire en considérant une pondération égalitaire entre les deux usagers. Sinon, il faut prendre la première valeur du détour donnant un gain proche de la borne donnée par un détour infini et qui se situe aux alentours de 10 minutes dans les cas que nous avons considérés.

Chapitre 5

Contributions et transferts industriels

La thèse a été l'occasion de contributions techniques et industrielles que nous présentons dans ce chapitre.
La section 5.1 présente le contexte de développement tant au niveau du laboratoire LAAS-CNRS que de la société MobiGIS. Puis dans la section 5.2 nous présentons la bibliothèque d'algorithmes développés pendant la thèse ainsi que son intégration dans les différents outils du laboratoire et de la société. Les principales contributions effectuées pour faire évoluer le calculateur d'itinéraires de la société sont détaillées dans la section 5.3. Enfin, la section 5.4 présente le travail effectué pour l'obtention d'une version en ligne du calculateur de la société : `MobiAnalyst SaaS/Cloud`.

5.1 Contexte de développement

5.1.1 Au laboratoire : la plate-forme *PlayMob'*[10]

La plate-forme *PlayMob'* est une plate-forme permettant d'intégrer, de valoriser et d'effectuer des démonstrations des différents travaux de recherche menés au laboratoire sur les problématiques de mobilité. Elle est constituée d'un ensemble de composants développés dans le cadre de stages et de thèses au LAAS-CNRS dans le but de fournir un cadre de développement d'algorithmes fonctionnant sur des graphes de transport réels.
Cette plate-forme regroupe un utilitaire de création de graphes de transport, un module de résolution contenant les différents algorithmes et une interface graphique dans un navigateur internet. Une capture d'écran de l'interface lors de l'utilisation du calculateur d'itinéraires point à point est donnée dans la figure 5.1.

L'utilitaire de création de graphe peut utiliser des données aux formats *OpenStreetMap* [9] pour la voirie et *GTFS* [11] pour les transports en commun, suivant la méthodologie exposée dans l'annexe A.
Les problèmes de mobilité initialement intégrés dans cette plateforme sont le calcul d'itinéraires point à point multimodal restreint ou non à un langage régulier (section 2.2.4) et la synchronisation des itinéraires de deux usagers dans le cadre du covoiturage (sous-section 2.3.2 et section 4.1).
Les primitives d'accès aux données ainsi que les algorithmes sont écrits en $C++$. La

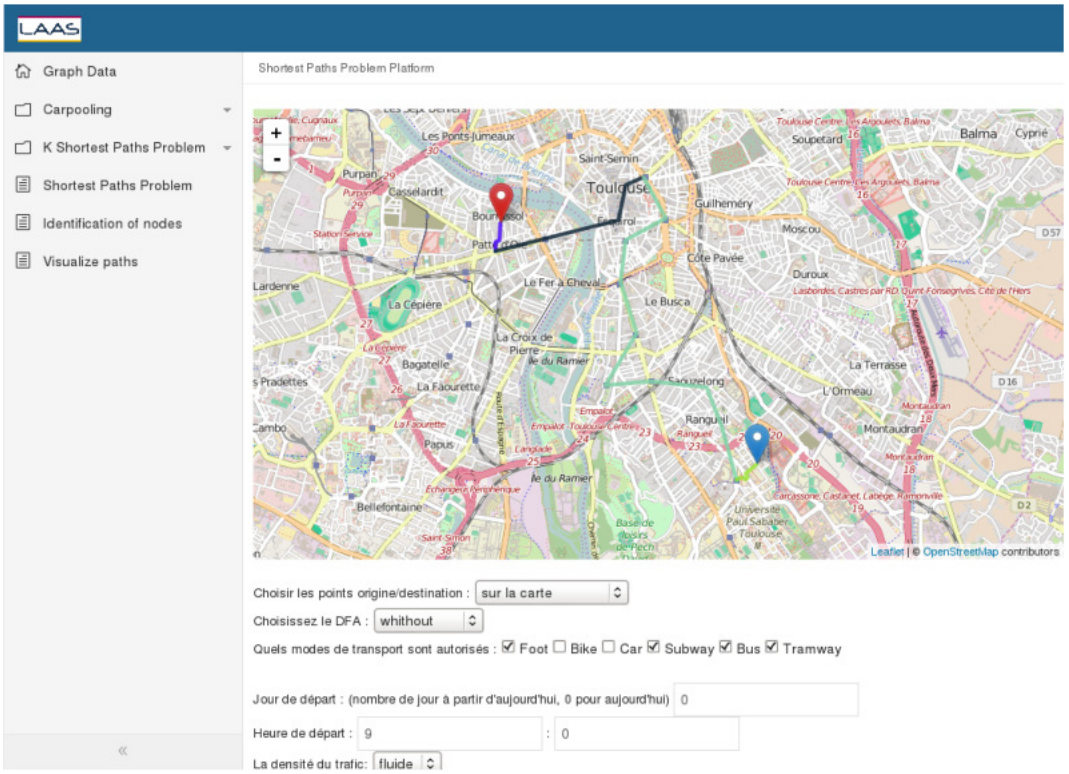


FIGURE 5.1 – Plate-forme *PlayMob'*

bibliothèque SWIG [12] permet d'accéder à ces fonctionnalités depuis *Python*, langage utilisé par le framework Django [3] utilisé pour l'interface graphique. Les visualisations cartographiques des données de transport et des résultats obtenus s'appuient sur le format GeoJSON [5] et la bibliothèque Leaflet [6].

5.1.2 Dans la société : le logiciel `MobiAnalyst` [7]

`MobiAnalyst` est un logiciel développé par la société MobiGIS. Il permet d'effectuer des analyses de réseaux de transport sur la base de calculateurs d'itinéraires et d'isochrones. `MobiAnalyst` se présente comme un extension du logiciel `ArcGIS` for Desktop de la suite `ArcGIS` [2], permettant la création, l'édition et l'utilisation de données géographiques spatialisées en environnement bureautique. La figure 5.2 montre le résultat d'un calcul d'isochrone dans `MobiAnalyst`.

De nombreux paramètres peuvent être pris en compte par le calculateur `MobiAnalyst` comprenant :

- le choix de la nature des coûts (i.e. impédance) à utiliser (temps ou distance) ;
- la sélection des différents modes de transport à considérer pour un calcul ;
- la sélection de profils piétons (pour simuler différents types de mobilité sur certains arcs du réseau de transport) ;
- la restriction ou non du temps total de marche ;
- la restriction du nombre de changements de mode de transport ;

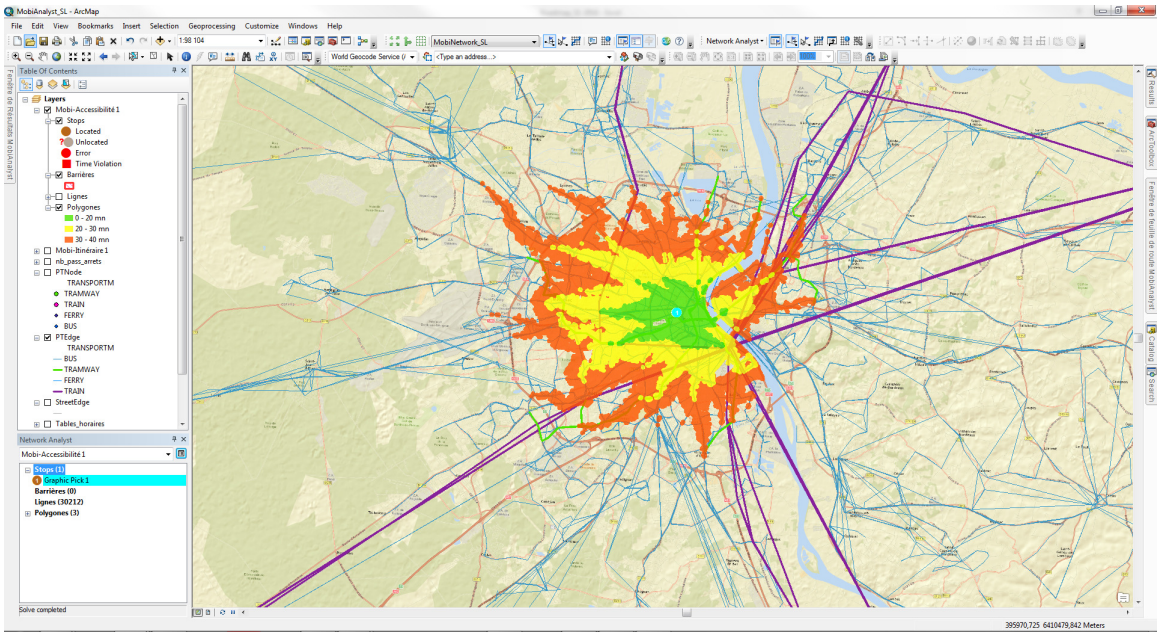


FIGURE 5.2 – Isochrone avec `MobiAnalyst`

— l'obligation de garer le véhicule et de terminer le trajet à pied et en transports en commun.

Il n'est pas possible de choisir un automate parmi une liste de choix possibles pour contraindre le trajet à un langage régulier, cependant, l'obtention de chemins passant obligatoirement par un parking se fait de façon interne par l'utilisation d'un automate. La sélection entre un automate acceptant tous les trajets et ceux contraignant un véhicule à stationner dans un parking avant la destination est effectuée de façon interne.
Une fois ces paramètres sélectionnés, `MobiAnalyst` se base sur un algorithme de calculs d'itinéraires multi-modaux et dépendant du temps bi-objectifs (en termes de temps de trajet et de nombre de transferts). Cet algorithme a été développé lors de précédents travaux de thèse [41]. Pour des raisons d'efficacité en termes de temps de calcul, la restriction du temps total de marche ou la limitation du nombre maximal de transfert sont traitées de manière heuristique dans le logiciel, le résultat obtenu étant alors sous-optimal en toute généralité. Cette sous-optimalité est visible lorsque l'on compare deux solutions issues d'un calcul de plus court chemin minimisant le temps et d'un autre minimisant la distance. Il se peut alors que l'un des trajets soit meilleur que l'autre à la fois pour le temps mais aussi pour la distance. En pratique, on a parfois observé, qu'un trajet minimisant la distance pourra aussi être plus rapide qu'un trajet minimisant le temps pour les mêmes origines et destinations.

Les résultats sont fournis sous forme de couches spatialisées, c'est à dire de données géographiques (lignes pour les itinéraires, polygones pour les isochrones) enrichies par des données attributaires (temps de trajet, nombre de modes utilisés, ...). Cela permet à la fois d'obtenir un résultat visuel mais aussi de disposer de données précises caractérisant

la solution. En outre, il est également possible d'effectuer des croisements avec d'autres données géographiques ; notamment en intersectant des isochrones avec des couches de population afin d'identifier le nombre de personnes concernées par une aire de desserte (par exemple le nombre de personnes habitant à moins de 2 minutes à pieds d'un arrêt de bus ou de métro).

5.2 Une bibliothèque de k plus courts chemins : *geroli*

5.2.1 Conception et développement

Le code produit durant la thèse, appartenant à la fois à la société MobiGIS et au CNRS, et pouvant être utilisé par les deux parties, a fait l'objet d'un développement autonome qui a conduit à la mise au point d'une bibliothèque appelée *geroli* pour Generic Routing Library
La bibliothèque *geroli* est écrite en $C++$, elle contient l'ensemble des algorithmes de calculs d'itinéraires développés pendant la thèse pour résoudre le problème des k plus courts chemins. Elle regroupe les algorithmes de Dijkstra, les adaptations proposées des algorithmes de Yen, REA ainsi que de l'algorithme IEA.
La bibliothèque est totalement détachée de l'implémentation des graphes. De cette façon les aspects multimodaux ainsi que la restriction des chemins à des langages réguliers n'est pas directement prise en compte par *geroli*. C'est lors de l'implémentation de la structure de graphe que l'utilisateur pourra simuler par une nouvelle structure de données le produit d'un graphe et d'un automate. Typiquement, ses nœuds seront des paires dont le premier élément correspond au nœud du graphe source et le second à l'état dans l'automate.

L'aspect primordial est que la bibliothèque *geroli* devait être adaptable à des environnements différents et convenir à la fois à la recherche et à la production. Comme mentionné ci-dessus, puisque le graphe est abstrait, l'adaptation de la bibliothèque à un environnement particulier se fait par la définition de fonctionnalités permettant d'y accéder. De la même façon, le traitement et l'enregistrement des données de la recherche est totalement autonome et peut donc être spécifique à chaque cas d'utilisation sans avoir à modifier les algorithmes. Cette fonctionalité est assurée par le concept de visiteurs qui ouvrent des points d'appel de fonction en différents endroits clés des algorithmes, passant en paramètres les informations disponibles localement dans la recherche afin que l'utilisateur puisse effectuer ses propres traitements de sauvegarde.

5.2.2 Intégration dans les différents environnements

Dans le cadre de la thèse, j'ai contribué au dévelopement de la plate-forme *PlayMob'*. Le travail a consisté principalement à effectuer la liaison entre *geroli* et la plate-forme afin de récupérer les algorithmes de k plus courts chemins dans l'interface graphique.

Pour ce faire, une structure permettant le produit d'un graphe de transport et d'un automate à la volée a été développée en se basant sur le code déjà existant mais qui n'était pas adaptable à *geroli*. On pourra noter que la recherche de chemins non soumis à une contrainte de langage régulier dispense de l'utilisation d'un automate. Il n'est cependant pas possible d'effectuer ce changement sans recompiler le code pour des raisons de performances. Plutôt que d'utiliser un automate acceptant tous les modes et comportant donc autant de transitions, une transition unique est utilisée afin de gagner un peu en performance. Ensuite, pour chaque algorithme, un simple emballage permettant d'abstraire les définitions propres à *geroli* permet aux développeurs *PlayMob'* d'appeler les fonctionnalités de cette bibliothèque de façon transparente depuis le $C++$ ou le *Python*. Ce travail a été effectué en collaboration avec une stagiaire de deuxième année à l'ISIMA.
L'image 5.3 témoigne de l'utilisation de *geroli* dans la plate-forme *PlayMob'* pour le calcul de k chemins. Ici, l'algorithme IEA est utilisé pour calculer les 50 premiers chemins élémentaires entre le Capitole et la basilique Saint-Sernin à Toulouse.

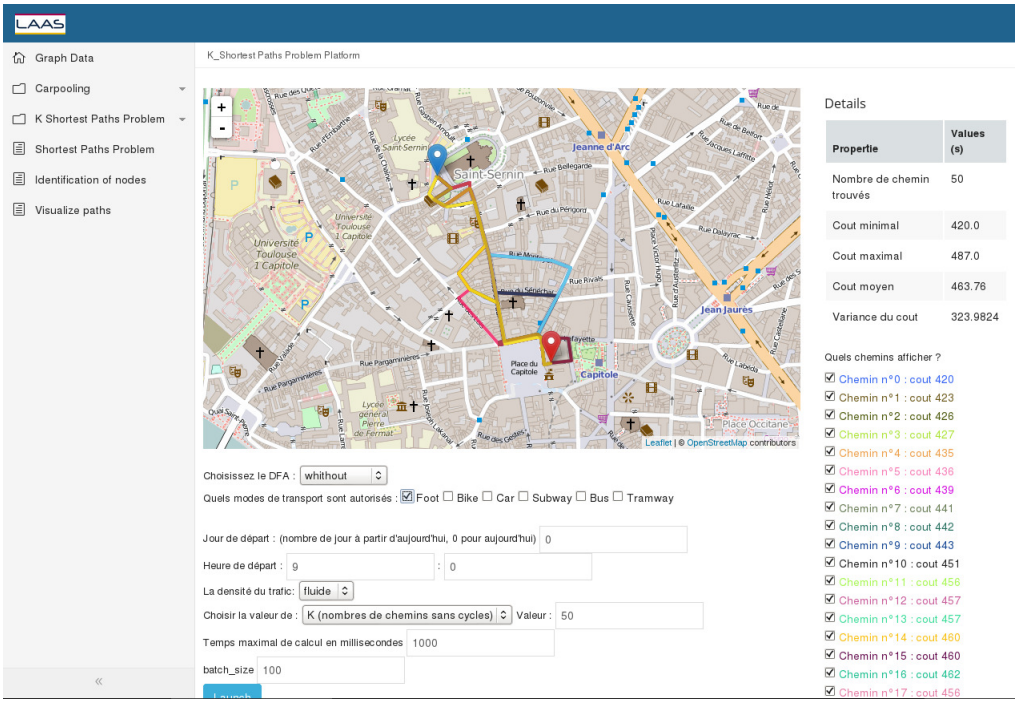


FIGURE 5.3 – Résultat d'un algorithme de k plus courts chemins sur la plate-forme *PlayMob'*

De la même façon, j'ai assisté un ingénieur de MobiGIS pour convertir un graphe `MobiAnalyst` en structure utilisable par *geroli*. Compte tenu de l'API proposée par `ArcGIS`, ce passage comporte du code assez particulier étant donné, par exemple, que le graphe utilisé par `ArcGIS` n'est pas dirigé mais comporte à chaque fois des informations concernant les deux sens de parcours, comme les restrictions et les temps de traversée.

5.3 Evolutions de MobiAnalyst

MobiAnalyst, dans sa forme bureautique comme extension de ArcGIS for Desktop, subit des évolutions ponctuelles en fonction d'une feuille de route produit et de besoins des clients de la société MobiGIS. J'ai contribué pendant ma thèse à certaines de ces évolutions. Celles représentant les plus grandes phases de travail sont détaillées ci-après.

5.3.1 Automatisation de la génération de graphes de transport

Dans MobiAnalyst, la création d'un graphe à partir du réseau de transport s'effectue en deux étapes principales. La première consiste à transformer les données sources en fichier shapefile, contenant les géométries ainsi que les champs associés à celles-ci, comme la longueur et le temps de parcours. La seconde utilise une API ArcGIS afin d'agréger ces différentes couches en un graphe, c'est la compilation. Lors de cette compilation, des attributs peuvent être spécifiés pour les arcs et pour les noeuds. La valeur que prend l'attribut dépend d'une fonction pouvant utiliser les champs de couches sources comme entrées, appelée évaluateur. Ainsi, on pourra récupérer la longueur géographique d'un arc par l'utilisation d'un attribut utilisant la géométrie de l'arc et par la suite, avoir un attribut donnant le temps de trajet piéton en introduisant une formule divisant la distance géométrique par une vitesse moyenne le long du type d'arc considéré.
Dans les versions précédentes de MobiAnalyst, la structure du graphe à compiler était codée en dur dans le programme. Il n'était donc pas possible de rajouter des attributs ou bien d'effectuer des variations dans leurs valuations sauf à modifier le code et fournir une nouvelle version spécifique.

Afin de palier ce problème, et dans le but de pouvoir générer des graphes de façon générique, permettant notamment de supporter plusieurs versions du logiciel MobiAnalyst avec un même outil configuré différement, un petit utilitaire spécifique a vu le jour. Il se base sur une description xml formelle du graphe à obtenir. Une liste donne l'ensemble des sources à prendre en compte, certaines pouvant être ignorées. De façon transversale, une liste d'évaluateurs détaille pour chaque attribut et pour chacune des sources à valuer un evaluateur particulier.

L'outil développé, appelé network builder, vise donc à produire un graphe compatible avec ArcGIS et MobiAnalyst à partir des différentes couches de données retravaillées. La construction des réseaux n'étant plus directement ouverte à tous les utilisateurs car trop particulière, l'outil network builder a été utilisé en production interne. Il a permis de générer des anciens et nouveaux réseaux, intégrant de la voirie uniquement, des transports en commun et pour certains des données trafic.

5.3.2 Ajout de données trafic

Lors de la création d'un réseau de transport, il est possible de spécifier pour chaque arc, le temps de parcours pour la voiture en fonction du type de tronçon ou de la vitesse maximale applicable correspondante. Cependant, la modélisation standard ne tient pas compte des fluctuations de la vitesse en fonction de la période et de l'heure de circulation.

L'intégration des données trafic historisées se fait en amont par l'incorporation de tables décrivant les vitesses de parcours des arcs. Il y a pour chaque arc disposant d'informations recueillies, un lien pour chaque jour de la semaine vers une table donnant un facteur de temps de parcours pour des périodes régulières de la journée, en général toutes les quinze minutes.
Ensuite, l'intégration dans le calculateur consiste à demander le temps de parcours lorsque l'on donne l'heure au départ de l'arc. Le récupération des valeurs de parcours étant faite par des fonctions d'`ArcGIS`, les modifications apportées au code ont été minimes. Il s'agissait donc principalement d'un travail lors de la compilation du graphe.

Le calculateur prend désormais en compte des données trafic basées sur des données historisées pour la voiture. La figure 5.4 présente le résultat d'un calcul d'itinéraire voiture sur un ensemble de données trafic, représentée par des couleurs indiquant la fluidité du trafic à l'heure du parcours. Ces données peuvent même être un flux temps réel, la gestion des données d'entrée, historisées ou temps réel, étant effectuée par `ArcGIS`.

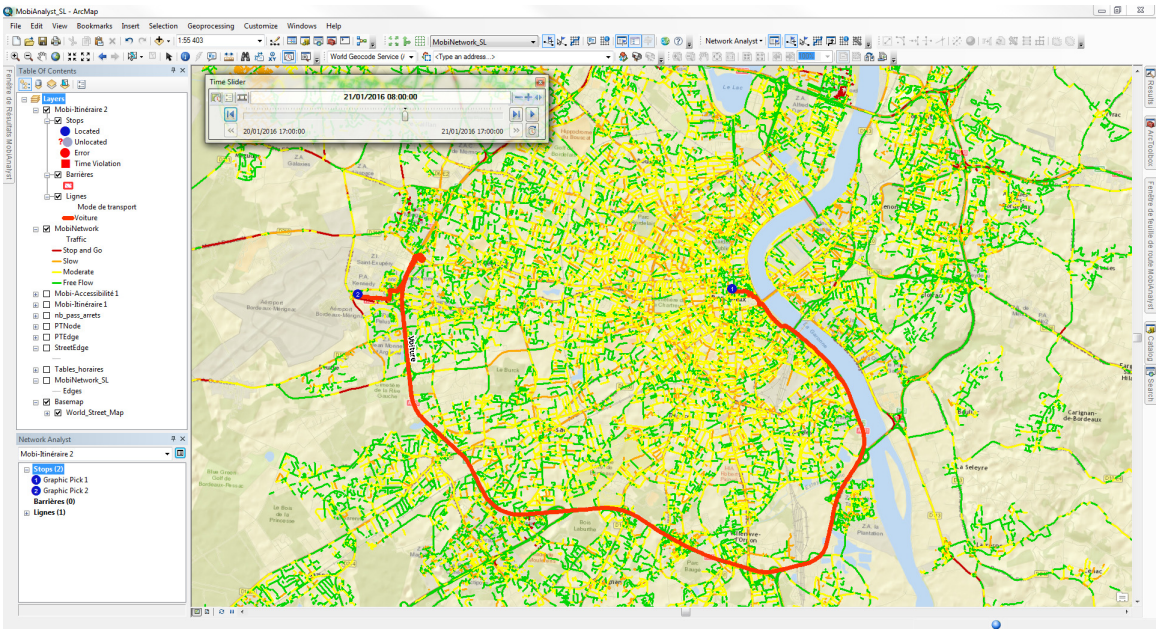


FIGURE 5.4 – Données trafic dans `MobiAnalyst`

5.3.3 Intégration de modes de transports en commun

Dans sa version initiale, `MobiAnalyst` ne prenait en compte que deux modes de transport en commun : le métro et le bus. Il était alors possible de contraindre l'usage de l'un comme de l'autre mais l'utilisateur ne pouvait pas prendre en compte d'autres modes de transports. Les autres modes de transport, comme par exemple le tramway, pouvaient être importés en les assimilant au métro, mais une fois le réseau de transport construit il n'était plus possible de différencier le mode métro correspondant à des lignes réelles de métro du mode métro correspondant en pratique à des lignes de tramway. Il n'était donc pas possible de contraindre de manière spécifique les modes ayant été assimilés au métro dans le modèle de réseau de transport. Afin de pouvoir prendre en compte de nombreuses situations de transport réelles, il était donc nécessaire de pouvoir intégrer dans `MobiAnalyst` l'ensemble des modes de transport existants.

Le travail a consisté à pouvoir importer plusieurs modes de transports, chacun comprenant des spécificités pouvant être un temps de transfert additionel pour tous les modes et des heures de début et fin de service ainsi qu'une fréquence pour les modes à fréquence. En amont, au lieu de créer une couche pour le métro et une couche pour le bus, une source unique est utilisée, comprenant l'ensemble des transports en commun. Un attribut supplémentaire permet alors d'identifier le transport associé à chaque arc et noeud. Ensuite, au niveau du code du calculateur, il fallait récupérer les valeurs spécifiques à chaque mode puis transférer tout ce qui était fait pour le bus (resp. le métro) pour tous les modes à horaires (resp. à fréquences) afin d'impacter le code existant le moins possible. La plus grosse partie revenait à récupérer les données liées à chaque mode puis à les exploiter en changeant le moins de code possible.

Les différents modes sont désormais regroupés sous trois familles de déplacements comprenant les modes statiques, comme la marche et la voiture, les modes à fréquences, comme les lignes de métro et les modes à horaires, comme les lignes de bus. On peut voir sur la figure 5.5 la fenêtre de paramètrage permettant de sélectionner indépendemment les différents modes de transport du réseau : bus, tram, train et ferry.

5.3.4 Mode console, python et export shape

Dans l'interface graphique `ArcGIS` for Desktop, les résultats de `MobiAnalyst` sont présents en tant qu'entités géométriques et attributaires, c'est à dire que l'on dispose des lignes résultats et d'informations pour chacune d'elle, telles que la distance, le temps, le temps cumulé, le temps d'attente, le mode courant, etc. Lorsque le nombre de calculs à faire est trop important deux problèmes surviennent :

— l'instance d'`ArcGIS` for Desktop reste inutilisable tant que le traitement n'est pas terminé ;

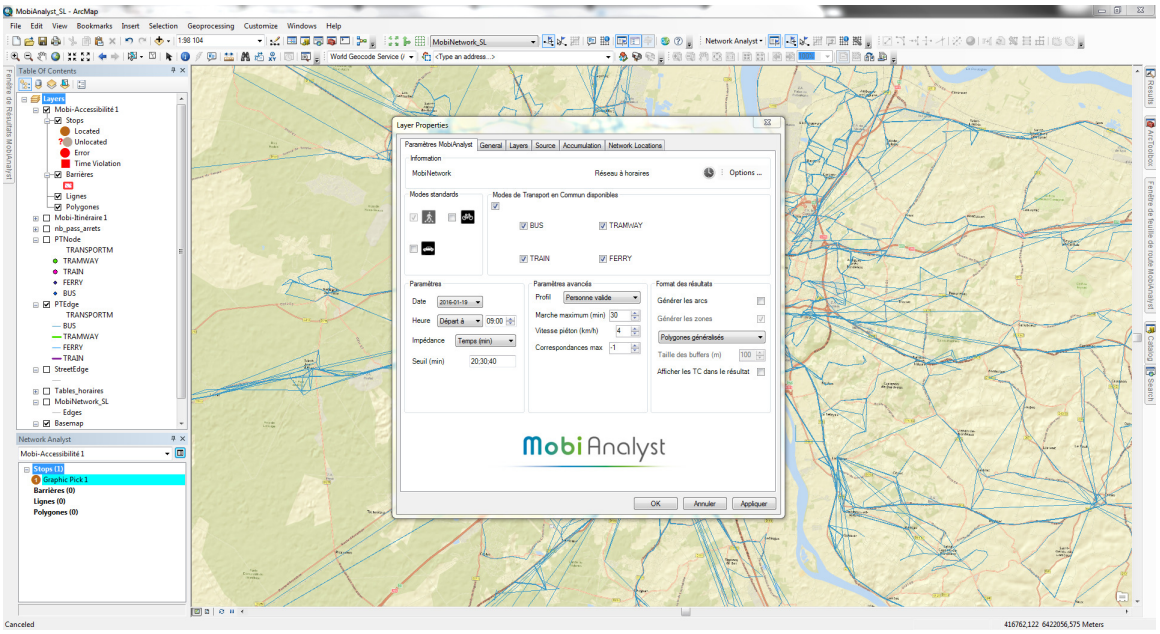


FIGURE 5.5 – Intégration des différents modes de transport dans MobiAnalyst

— les résultats géométriques ne sont pas exploitables car trop nombreux. Ils se superposent les uns aux autres et il semble inenvisageable de tous les consulter unitairement.

Une première version du mode console consistait à fournir les informations attributaires sans la géométrie associée, sous la forme d'un fichier csv. En effet, bien souvent, lors de calculs massifs (tels que des calculs de distancier), la réponse géographique est subsidiaire et l'interêt se porte principalement sur les valeurs de parcours afin de déterminer des données statistiques sur l'échantillon examiné.
Il a toutefois été nécessaire de sauvegarder les géométries de tous les résultats. Pour cela, il a fallu récupérer un grand nombre de fonctionnalités présentes dans une classe attachée au calculateur présent dans l'interface. Cette classe n'étant pas statique, il a fallu déplacer les portions de code appropriées dans une zone ne dépendant pas de cet environnement pour pouvoir les appeler à la fois depuis la console mais aussi depuis le solveur de l'interface graphique.

Le mode console permet désormais d'appeler le calculateur sans avoir besoin de l'interface graphique d'ArcGIS for Desktop, ce qui permet à des clients de lancer des calculs lourds sur plusieurs heures, par exemple lorsqu'il s'agit de calculer des matrices origine-destination avec de nombreux points. De façon pratique, une API python appelant le mode console permet aux utilisateurs de chaîner plusieurs traitements automatiques avant et après le calcul.

5.3.5 Maintenances et performances

Comme pour de nombreux logiciels, la correction d'erreurs prend une part non négligeable dans le processus de production. Suite à des retours utilisateurs, de nombreux légers problèmes ont été corrigés dans `MobiAnalyst`. Ce travail a été réalisé en partie avec un stagiaire de quatrième année de l'INSA de Toulouse.

Le calculateur de `MobiAnalyst` souffrait en production de performances moyennes. Ces performances ont pu être améliorées en travaillant sur l'accès aux tables horaires stockées dans une base de données SQLite, principalement par la modification des index de recherche. Ces améliorations de performance ont été réalisées en collaboration avec un stagiaire de Master 2 Recherche en Informatique, parcours Recherche Opérationnelle de l'Universite Paul Sabatier, qui a mené une campagne de tests de performance à la suite de ces modifications.

5.4 `MobiAnalyst SaaS/Cloud`

L'utilisation de services en ligne se répandant de plus en plus, la version de `MobiAnalyst` desktop a été portée sur le cloud en s'appuyant sur les connaissances existantes tant au niveau du solveur qu'en ce qui concerne le fonctionnement des bibliothèques fournies par `ArcGIS`.

Dans sa version bureautique, le solveur `MobiAnalyst` écrit en $C++$ s'interfaçait avec du code $C\#$ en utilisant une méthode appellée le «marshalling» pour passer des structures de données de l'un à l'autre. L'inconvénient de cette méthode réside dans le fait que l'algorithme ne peut pas directement écrire les résultats dans les objets de l'interface mais est obligé de passer par une sérialization en structures pour finalement reconstruire les résultats dans l'interface en $C\#$. Cela induit donc un coût mémoire et temps qui pourrait être totalement supprimé.
Un problème supplémentaire de l'utilisation du $C\#$ pour communiquer avec l'API `ArcGIS`, ne contenant au final que des objets $C++$, est que chaque appel de fonction induit un coût supplémentaire lié à la redirection vers une fonction $C++$ à travers le serveur `COM`. Lorsque le nombre d'appels devient important, par exemple lorsqu'ils sont faits dans des boucles imbriquées, ce qui était le cas pour construire les polygones des isochrones, ce coût devient prohibitif. C'est d'ailleurs indiqué dans la documentation de l'API $C\#$ de `ArcGIS`.

Pour palier ces différents problèmes, le passage en mode serveur de `MobiAnalyst` a été l'occasion d'une refonte totale du solveur. La partie basse contenant les algorithmes, l'accès au données et la construction des résultats, est entièrement écrite en $C++$. Une couche, obligatoirement écrite $C\#$ en raison de l'architecture des serveurs `ArcGIS`,

prend en charge les différents graphes mis à disposition des utilisateurs et appelle la partie basse. Une dernière en *Java* gère les droits d'accès ainsi que les coûts d'utilisation du service pour chaque utilisateur. Mon travail s'est focalisé sur la partie basse du service. Pour les algorithmes, la bibliothèque *geroli*, étendue de quelques composants de spécialisation pour les besoins de l'application (accès au graphe, construction des résultats), est utilisée pour les itinéraires points à points, les distanciers et les isochrones. Cette industrialisation a permis de vérifier que le modèle programmatique expérimenté dans *geroli* était suffisament souple et performant pour être utilisé en production. En effet, la construction des résultats est bien plus complexe que ce qui est fait dans *geroli* pour des graphes aléatoires ou *PlayMob'*, puisqu'elle suppose la manipulation d'objets permettant d'écrire dans des tables attributaires et d'effectuer des opérations géométriques.
De façon technique, ces travaux ont necessité une montée en compétence sur la technologie `COM` (*Component Object Model*) qui permet de définir et d'implémenter une interface pouvant être appelée par d'autres programmes et langages par le biais d'un serveur de stockage et d'appel des différents composants. Son usage est facilité par l'utilisation de la bibliothèque `ATL` (*Active Template Library*) fournissant, par l'intermédiaire de templates, un certain nombre de fonctionnalités redondantes à écrire. Cependant, la programmation générique en utilisant les objets `COM` n'est pas très évidente dès que l'on sort de l'utilisation prévue de cette bibliothèque. Par exemple, la gestion des exceptions, indispensable à l'utilisateur final qui ne souhaite pas qu'on lui dise seulement qu'une erreur est survenue mais qui attend une description de l'erreur, s'avère délicate puisque l'interfacage `COM` est imperméable aux exceptions. C'est à dire qu'une exception $C++$ standard si elle termine l'appel `COM` n'est pas récupérée par l'appelant. Pour qu'une méthode `COM` d'une classe concrète puisse lever une exception, il faut qu'elle implémente une fonction tierce ($InterfaceSupportsErrorInfo$) faisant partie d'une interface dédiée ($ISupportErrorInfo$), qui détermine si cette classe gère les erreurs pour l'interface qui requiert la méthode appelée. La solution a permis de factoriser le code de la fonction tierce pour toutes les classes implémentées et simplifie l'utilisation des exceptions. Le code en $C++$ peut envoyer une exception qui sera ensuite décodée par une fonction de la classe générique, puis transmise au serveur `COM` comme erreur compréhensible. Cette méthode a tout de même pour inconvénient de dupliquer du code mais cette redondance de code est très basique. Il convient en fait de copier-coller quelques lignes de code pour encapsuler chaque fonction `COM` dans l'environnement gérant les exceptions, tout en évitant de dupliquer le code gérant ces exceptions.

Le service est désormais opérationel, comportant de nombreuses fonctionnalités développées par des ingénieurs à MobiGIS et il continue d'être amélioré de façon permanente afin de combler le retard fonctionnel par rapport à la version antérieure. Les images 5.6 et 5.7 affichent les résultats respectifs d'un calcul d'isochrone et d'itinéraire sur l'interface de démonstration de `MobiAnalyst SaaS/Cloud`.

De façon pratique, les performances du nouveau solveur sont meilleures que celles de `MobiAnalyst`, ce qui constitue un point important d'amélioration. Il bénéficie, d'autre part, d'un modèle plus évolutif et j'espère plus souple face aux futures attentes qui seront formulées.

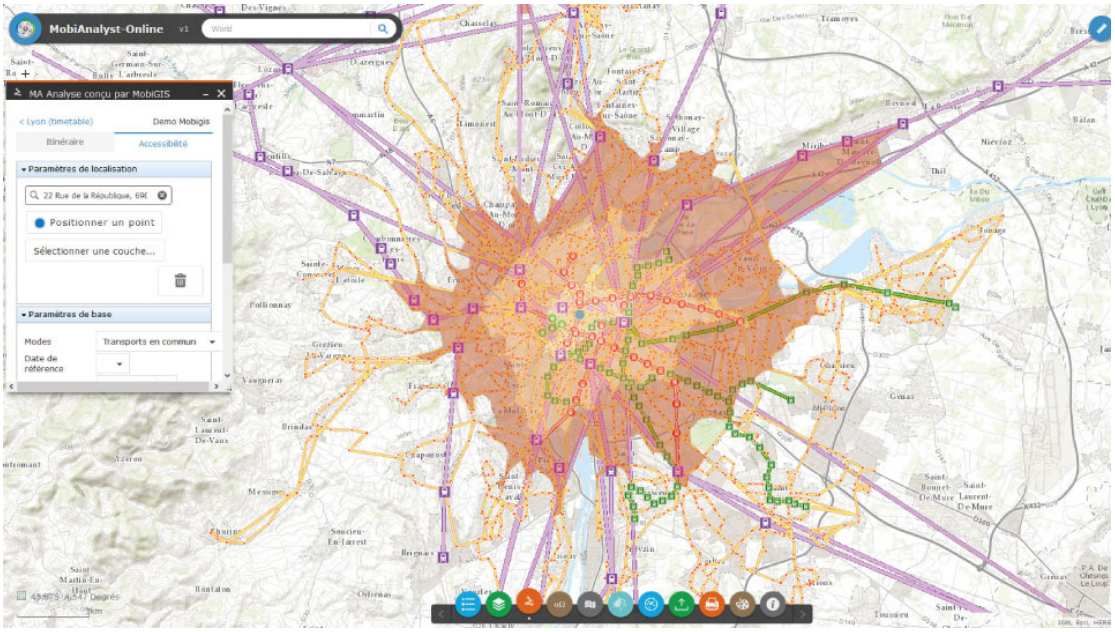


FIGURE 5.6 – Résultat d'un calcul d'isochrone dans `MobiAnalyst SaaS/Cloud`

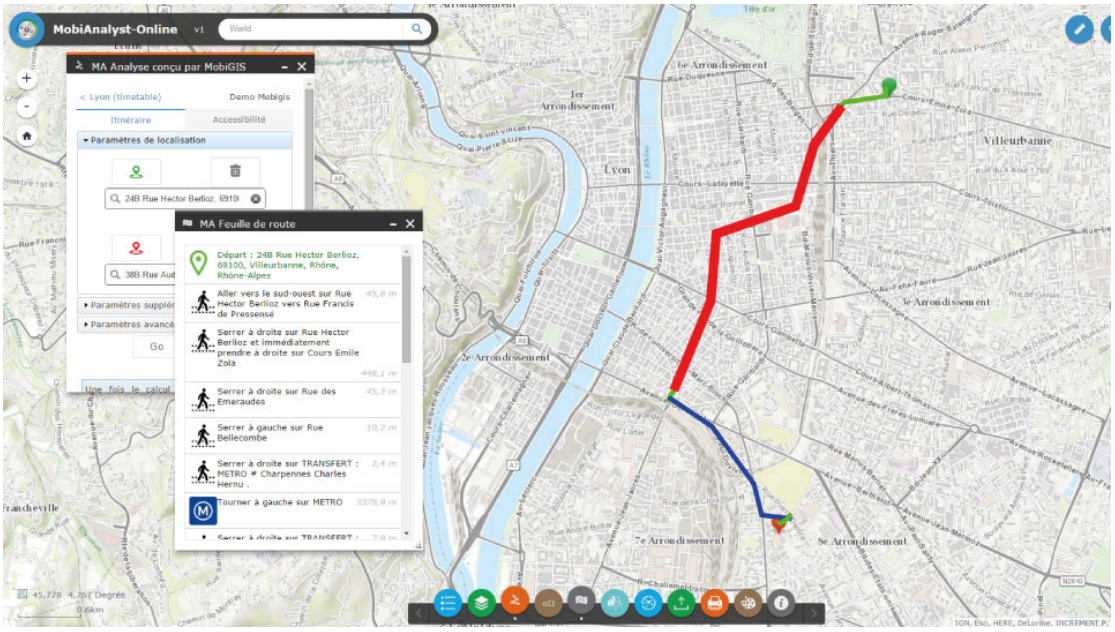


FIGURE 5.7 – Résultat d'un calcul d'itinéraire et sa feuille de route dans `MobiAnalyst SaaS/Cloud`

5.5 Covoiturage et Moveazy

En complément des calculateurs d'itinéraires, MobiGIS développe des solutions de covoiturage. Une preuve de concept liée au projet ANR Amores [1] a permis, dans une phase initiale, d'aborder l'aspect théorique de la résolution du 2SPSPP et l'aspect pratique de la mise en relation des candidats potentiels. Par la suite, l'application Moveazy a été developpée par la société MobiGIS pour une utilisation par le grand public.

Dans le cadre du projet Amores, la société avait à charge de développer une interface logicielle de calcul de covoiturage dynamique pouvant être intégrée avec des composants liés à la sécurité et à la protection des données personnelles comme la géolocalisation ou des services de notation des usagers. L'équipe de développement a implémenté l'algorithme 2SPSPP en *Java* en se basant sur l'article |Arthur| et j'ai pour ma part codé la partie qui gère les automates pour prendre en compte la restriction à des chemins réguliers. Cela comprenait la structure et la création des automates à partir d'expressions régulières, l'inversion des automates afin de gérer le parcours inversé ainsi que leur déterminisation pour appliquer les algorithmes sur des automates déterminisés.
Le service mis en place pour Moveazy se calque sur celui de MobiAnalyst SaaS/Cloud et retourne pour une paire d'origines et de destinations les points de rendez vous optimaux et les parcours associés, l'appariement des différents utilisateurs étant fait par une couche plus haute du service. J'ai principalement supervisé un stagiaire du Master 2 Recherche en Informatique, parcours Recherche Opérationnelle de l'Universite Paul Sabatier, qui a implémenté l'algorithme 2SPSPP spéfiquement pour l'environement ArcGIS.

Le service de covoiturage est maintenant disponible dans l'application Moveazy, il permet de calculer des itinéraires pour des usagers à pied ou en voiture mais également d'offrir et de demander un service de covoiturage. Après un pré-calcul appelant le nouveau service déterminant les points de rendez-vous optimaux pour des paires de participants, les utilisateurs sont ensuite mis en relation en fonction de leur positions géographiques, de leur préférences et de leurs heures de trajet. La figure 5.8 montre la sélection d'un trajet parmis plusieurs propositions de covoiturage dans l'application Moveazy.

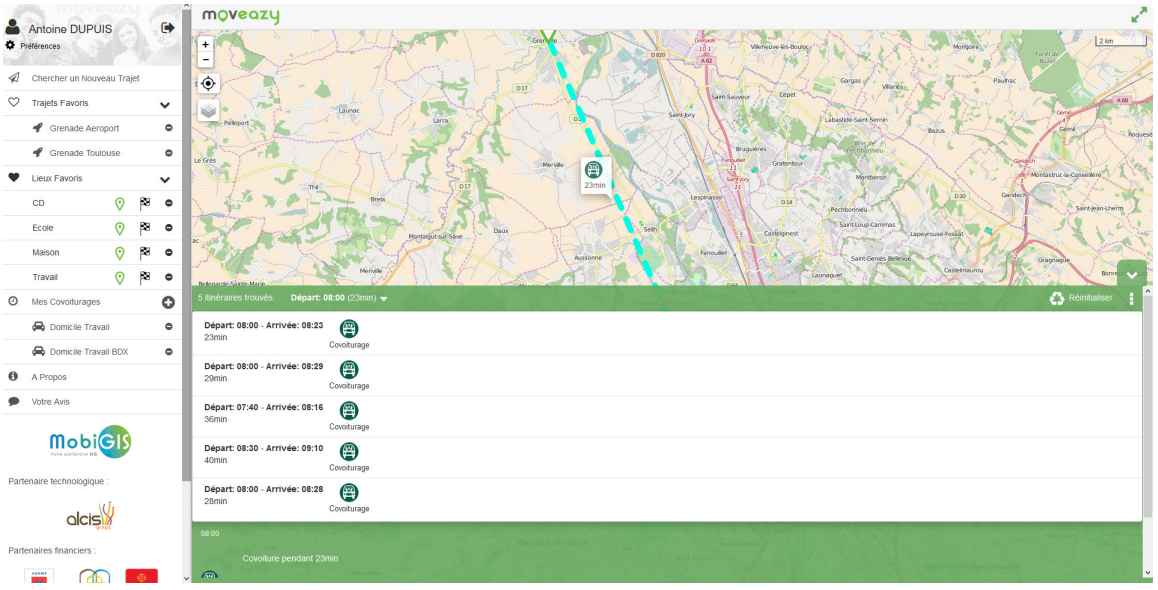


FIGURE 5.8 – Une possibilité de trajet parmis cinq choix de covoiturages proposés à l'utilisateur de la plate-forme `Moveazy`

Chapitre 6

Conclusions et Perspectives

Il peut sembler au lecteur que le calcul de plus courts chemins est si bien établi qu'il n'a plus vocation à des activités de recherche. C'est également l'impression que j'en avais eue au commencement de la thèse et j'espère que la lecture de ce manuscrit a modestement contribué à démontrer le contraire. Premièrement parce que comme présenté dans l'état de l'art, les méthodes de résolution ne cessent de s'améliorer, du moins sur le plan expérimental à défaut de perfectionnements théoriques. Ensuite par le fait que de nouvelles thématiques stimulent la recherche par l'ajout de difficultés provenant des données (graphes multimodaux, dépendance au temps, prise en compte de plusieurs objectifs, ...) et de la combinatoire des problèmes (contrainte de langage, k plus courts chemins, covoiturage). Enfin parce que la modélisation de problèmes s'y rapportant et l'implémentation de leur solution est une interrogation fréquente en informatique et qu'aucune implémentation ne fait vraiment référence car cela dépend des particularités des problèmes ainsi que des contraintes opérationnelles qui leur sont associées.

Le travail présenté s'est déroulé dans le cadre d'une thèse CIFRE pour l'entreprise MobiGIS. De nombreuses implications techniques, notamment en ce qui concerne le portage de `MobiAnalyst` pour l'internet, et plus ou moins éloignées des activités de recherche ont toutefois pu bénéficier du travail effectué en laboratoire. La description et la programmation des algorithmes, l'intégration des contraintes industrielles de développement et de fonctionnement, la compréhension et l'affinement des méthodes de résolution de problèmes de plus courts chemins est prégnante au cœur de ces nouveaux logiciels.

La première thématique abordée dans cette thèse concerne le calcul des k plus courts chemins pour des réseaux de transport multimodaux et dépendant du temps et l'exploitation de l'ensemble de ces chemins pour caractériser des trajets alternatifs. Les travaux menés ont permis d'étendre, pour le contexte des réseaux de transport, un algorithme de calcul de k plus courts chemins élémentaires (extension de l'algorithme de Yen) et un algorithme de calcul de k plus courts chemins avec cycles (extension de l'algorithme REA). Nous avons également proposé un nouvel algorithme déterminant les k plus courts chemins avec cycles (algorithme IEA). Afin d'obtenir des chemins viables pour un usager, certains chemins avec cycles sont éliminés, soit a posteriori, soit en introduisant des procédures de coupe cycle (à l'origine du trajet ou avec une taille limitée). Les expérimentations, menées sur le réseau réel de l'agglomération toulousaine ont montré que les

temps de calcul pour l'algorithme de Yen deviennent rapidement prohibitifs en fonction du nombre de chemins demandé. Concernant les algorithmes REA et IEA, calculant des chemins avec cycle, les temps de calcul restent raisonnables mais l'espace mémoire utilisé devient vite important pour l'algorithme IEA et dans une moindre mesure pour l'algorithme REA. Ainsi, l'algorithme IEA agrémenté d'une procédure coupant les cycles de longueur 2 fournit k plus courts chemins de manière plus rapide que l'algorithme de Yen, certes dans un ordre heuristique mais très proche de l'optimum. La démonstration qu'une telle méthode de calcul des k plus courts chemins avec cycles concurrence les algorithmes de calcul de chemins élémentaires est un résultat important qui à notre connaissance n'avait jamais été mis en évidence.
L'efficacité d'une telle procédure permet de générer un grand nombre de chemins afin qu'ils soient sélectionnés comme des alternatives par un algorithme générique de distinction des chemins faisant appel à un critère de différenciation. Deux types de critères de différenciation ont été retenus mais il semble difficile de caractériser une bonne mesure de la différence entre deux chemins dans un cadre général. D'où l'intérêt d'une approche en deux temps : génération de chemins puis filtrage, qui est flexible et adaptable car permettant de changer en fonction des préférences de l'utilisateur ou de l'usager une procédure de filtrage adaptée, la génération de chemins restant inchangée.
Concernant cette thématique de travail, une première perspective consisterait à compléter les expérimentations par des réseaux de transport de taille et de densité différentes afin de mieux caractériser l'efficacité des méthodes de calcul des k plus courts chemins. Un travail complémentaire serait à mener pour obtenir d'autres méthodes de différenciation des chemins, par exemple en utilisant des méthodes issues de l'analyse de données comme les k-moyennes. Il serait par ailleurs intéressant d'intégrer les procédures de calculs de chemins et les procédures de sélection d'alternatives afin de pouvoir relancer des calculs de chemins si le nombre d'alternatives obtenues n'est pas satisfaisant. Enfin, de tels calculs pourraient être menés sur un modèle de réseau de transport centré sur les transports en commun, à l'image du modèle utilisé pour le développement de l'algorithme `raptor` [33] afin de calculer des plus courts chemins multi-objectifs.

La seconde thématique considérée dans cette thèse concerne le problème de calcul de points de synchronisation entre deux usagers dans le cadre du covoiturage dynamique, problème appelé `2SPSPP`. Nous avons tout d'abord analysé l'impact des variations de la dépendance au temps des trajets pour le conducteur ou le passager. Le résultat de cette analyse est que l'approche développée dans la littérature pour résoudre le `2SPSPP` est très performante et adaptative. Et nous avons proposé un algorithme générique permettant de gérer ces combinaisons. Cependant, il reste quelques combinaisons, en particulier lorsque le trajet du conducteur dépend des horaires, pour lesquelles cette approche ne permet pas d'obtenir la solution optimale.
Nous avons étudié le cas de plusieurs passagers se synchronisant sur un point de ren-

contre et un point de séparation communs à tous. L'approche issue du 2SPSPP peut être appliquée à plusieurs situations, mais lorsque les passagers ont des destinations différentes et des temps de trajet dépendant des horaires, elle ne fournit plus la solution optimale. Nous nous sommes intéressés à l'impact des horaires de départ ou d'arrivée pour les usagers. Dans le cas de l'inversion du problème 2SPSPP pour résoudre un problème de synchronisation avec heures de départ au plus tard, nous avons formulé une règle de dominance optimale et une règle de dominance heuristique permettant le filtrage des étiquettes dans le dernier algorithme pour le piéton. Les analyses expérimentales ont montré que les résultats sont parfaitement symétriques suivant que les horaires soient donnés aux origines ou aux destinations. Pour le problème combinant une heure de départ pour un usager et une heure d'arrivée pour le second usager, nous avons mis en évidence des problèmes spécifiques liés à la cohérence des deux horaires (arrivée au plus tôt et départ au plus tard) générés au niveau du dernier algorithme concernant le piéton.

Enfin, le problème aggrémenté d'une limitation du détour effectué par le conducteur pour prendre en charge le passager a été présenté et un algorithme optimal dédié basé sur un pré-traitement et un filtrage des sommets ne respectant pas la condition de détour a été proposé. L'analyse expérimentale révèle qu'il existe un seuil d'une dizaine de minutes sur les cas que nous avons considérés et à partir duquel le système global passager-conducteur devient gagnant, tout en garantissant un temps de calcul inférieur à la seconde, un détour effectif de la voiture modéré par rapport au détour maximum autorisé. Cependant, pour une valeur de détour maximal plus importante, les gains n'augmentent presque plus si bien qu'il devient improbable que le conducteur consente à effectuer une telle déviation.

Ainsi dans cette seconde thématique, nous avons pu analyser différents cas d'application de l'approche développée pour résoudre le 2SPSPP. Cette analyse a permis de mettre en évidence les limites de l'approche en particulier en présence de différents trajets dépendant du temps. Dans ces situations, une autre approche de résolution serait à développer, en se basant par exemple sur des approximations des trajets dépendant des horaires couplées à des ré-évaluations ainsi que sur des calculs de bornes afin de limiter le temps de calcul.

Les cas de covoiturage dynamique considérés supposent qu'un appariement conducteur-passager a déjà été effectué. Une extension de ces travaux pourrait être de considérer de manière intégrée les calculs d'appariement et de trajets. Pour cela, la qualité d'un appariement pourrait être estimée par la qualité du trajet obtenu. Néanmoins l'approche énumérative de l'ensemble des appariements possibles serait trop coûteuse en termes de temps de calcul et pourrait s'appuyer sur des évaluations heuristiques des trajets communs afin de permettre une mise en œuvre dans un contexte dynamique ou temps réel. Une autre extension considérerait plusieurs passagers avec des points de synchronisation d'itinéraires différents. Une des approches possibles serait alors de considérer le pro-

blème comme séquentiel : de calculer incrémentalement le trajet d'un conducteur avec un ensemble de piétons avant d'y ajouter le suivant. Dans tous les cas, de nombreuses perspectives seront à envisager en fonction des usages réels effectués par les acteurs du covoiturage. C'est donc un nouvel ensemble de problématiques, pouvant être résolues par des outils connus ou restant à inventer, qui a commencé et continuera de se poser à la recherche.

Bibliographie

[1] Amores. http://amores-project.org/. (Cité en page 105.)

[2] ArcGIS. http://www.arcgis.com. (Cité en page 94.)

[3] Django. https://www.djangoproject.com/. (Cité en page 94.)

[4] Geofabrik. http://www.geofabrik.de. (Cité en page 44.)

[5] Geojson. http://geojson.org/. (Cité en page 94.)

[6] Leaflet. http://leafletjs.com/. (Cité en page 94.)

[7] MobiAnalyst. http://www.mobianalyst.fr. (Cité en pages vi et 94.)

[8] Neptune. http://www.normes-donnees-tc.org/category/neptune/. (Cité en page 118.)

[9] OpenStreetMap. http://www.openstreetmap.org. (Cité en page 93.)

[10] PlayMob'. http://www.laas.fr. (Cité en pages vi et 93.)

[11] GTFS. https://developers.google.com/transit/gtfs/reference. (Cité en pages 93 et 118.)

[12] SWIG. http://www.swig.org. (Cité en page 94.)

[13] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Alternative routes in road networks. J. Exp. Algorithmics, 18 :1.3 :1.1–1.3 :1.17, April 2013. (Cité en page 20.)

[14] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing : A review. European Journal of Operational Research, 223(2) :295 – 303, 2012. (Cité en pages 16 et 55.)

[15] Kamel Aissat and Ammar Oulamara. Computational Logistics : 6th International Conference, ICCL 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings, chapter The Round-Trip Ridesharing Problem with Relay Stations, pages 16–30. Springer International Publishing, Cham, 2015. (Cité en page 17.)

[16] Kamel Aissat and Sacha Varone. Enterprise Information Systems : 17th International Conference, ICEIS 2015, Barcelona, Spain, April 27-30, 2015, Revised Selected Papers, chapter Carpooling as Complement to Multi-modal Transportation, pages 236–255. Springer International Publishing, Cham, 2015. (Cité en page 18.)

[17] Husain Aljazzar and Stefan Leue. K* : A heuristic search algorithm for finding the k shortest paths. Artificial Intelligence, 175(18) :2129 – 2154, 2011. (Cité en page 27.)

[18] Christian Artigues, Marie-José Huguet, Fallou Gueye, Frédéric Schettini, and Laurent Dezou. State-based accelerations and bidirectional search for bi-objective multi-modal shortest paths. Transportation Research Part C : Emerging Technologies, 27 :233 – 259, 2013. Selected papers from the Seventh Triennial Symposium on Transportation Analysis (TRISTAN VII). (Cité en pages 1 et 20.)

[19] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In Alberto Marchetti-Spaccamela and Michael Segal, editors, Proceedings of the First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems, TAPAS'11, pages 21–32, Berlin, Heidelberg, 2011. Springer-Verlag. (Cité en page 21.)

[20] Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav Marathe, and Dorothea Wagner. Engineering Label-Constrained Shortest-Path Algorithms, volume 5034 of AAIM '08, chapter 5, pages 27–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. (Cité en page 15.)

[21] Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. SIAM J. Comput., 30(3) :809–837, May 2000. (Cité en page 15.)

[22] Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result diversity for multi-modal route planning. In Daniele Frigioni and Sebastian Stiller, editors, ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, volume 33 of OpenAccess Series in Informatics (OASIcs), pages 123–136, Sophia Antipolis, France, September 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cité en page 20.)

[23] Reinhard Bauer and Daniel Delling. Sharc : Fast and robust unidirectional routing. J. Exp. Algorithmics, 14 :4 :2.4–4 :2.29, January 2009. (Cité en page 13.)

[24] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, September 1954. (Cité en page 8.)

[25] Richard Bellman. On a routing problem. Quarterly of Applied Mathematics, 16 :87–90, 1958. (Cité en page 8.)

[26] Arthur Bit-Monnot, Christian Artigues, Marie-José Huguet, and Marc-Olivier Killijian. Carpooling : the 2 synchronization points shortest paths problem. In 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS), page 12 p, Sophia Antipolis, France, September 2013. (Cité en pages 1, 17, 55, 56, 57, 62, 63, 64 et 90.)

[27] Aurelie Bousquet, Sophie Constans, and El Faouzi Nour-Eddin. On the adaptation of a label-setting shortest path algorithm for one-way and two-way routing in multimodal urban transport networks. INOC, pages 1–8, April 2009. (Cité en page 16.)

[28] Y. Chen, M. G. H. Bell, and K. Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. IEEE

Transactions on Intelligent Transportation Systems, 8(1) :14–20, March 2007. (Cité en pages 21 et 47.)

[29] Noam Chomsky. Three models for the description of language. Information Theory, IRE Transactions on, 2(3) :113–124, 1956. (Cité en page 121.)

[30] B. C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, MIT, May 1999. (Cité en page 14.)

[31] B. C. Dean. Shortest paths in fifo time-dependent networks : Theory and algorithms. Technical report, MIT, 2004. (Cité en page 14.)

[32] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Robust and Online Large-Scale Optimization : Models and Techniques for Transportation Systems, chapter Engineering Time-Expanded Graphs for Faster Timetable InformationFinding the K Shor, pages 182–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. (Cité en page 14.)

[33] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. Transportation Science, 49(3) :591–604, 2015. (Cité en pages 20 et 108.)

[34] Edsger W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1(1) :269–271, 1959. (Cité en page 8.)

[35] D. Eppstein. Finding the k shortest paths. Technical Report 94-26, University of California, Irvine, May 1994. (Cité en page 22.)

[36] David Eppstein. Finding the k shortest paths. SIAM Journal on Computing, 28(2) :652–673, 1998. (Cité en pages 22 et 27.)

[37] Lester Randolph Ford. Network flow theory. Technical report, RAND, August 1956. (Cité en page 8.)

[38] G.N. Frederickson. An optimal algorithm for selection in a min-heap. Information and Computation, 104(2) :197 – 214, 1993. (Cité en page 27.)

[39] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path : A* search meets graph theory. In Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. (Cité en page 12.)

[40] Tristram Gräbener. Calcul d'itinéraire multimodal et multiobjectif en milieu urbain. These, Université des Sciences Sociales - Toulouse I, November 2010. (Cité en page 19.)

[41] Fallou Gueye. Algorithmes de recherche d'itinéraires en transport multimodal. These, Université de Toulouse - INSA, December 2010. (Cité en pages 1 et 95.)

[42] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on, 4(2) :100–107, February 1968. (Cité en page 10.)

[43] Marie-José Huguet, Dominik Kirchler, Pierre Parent, and Roberto Wolfler Calvo. Efficient algorithms for the 2-Way Multi Modal Shortest Path Problem. In International Network Optimization Conference (INOC), volume 41, pages 431 – 437, Tenerife, Spain, May 2013. Elsevier. 7. (Cité en pages 16 et 17.)

[44] Takahiro Ikeda, Min-Yao Hsu, Hiroshi Imai, Shigeki Nishimura, Hiroshi Shimoura, Takeo Hashimoto, Kenji Tenmoku, and Kunihikoqcd Mitoh. A fast algorithm for finding better routes by ai search techniques. In Vehicle Navigation and Information Systems Conference, pages 291–296, 1994. (Cité en page 11.)

[45] Víctor M. Jiménez and Andrés Marzal. Algorithm Engineering : 3rd International Workshop, WAE'99 London, UK, July 19–21, 1999 Proceedings, chapter Computing the K Shortest Paths : A New Algorithm and an Experimental Comparison, pages 15–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. (Cité en page 29.)

[46] Víctor M. Jiménez and Andrés Marzal. A lazy version of eppstein's k shortest paths algorithm. In In : WEA, pages 179–190. Springer, 2003. (Cité en page 27.)

[47] Dominik Kirchler, Leo Liberti, Thomas Pajor, and Roberto Wolfler Calvo. Unialt for regular language contrained shortest paths on a multi-modal transportation network. In Alberto Caprara and Spyros Kontogiannis, editors, 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, volume 20 of OpenAccess Series in Informatics (OASIcs), pages 64–75, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cité en page 16.)

[48] Ulrich Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung, volume 22, pages 219–230, 2004. (Cité en page 13.)

[49] Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science, 18(7) :401–405, 1972. (Cité en page 25.)

[50] Angelica Lozano and Giovanni Storchi. Shortest viable path algorithm in multimodal networks. Transportation Research Part A : Policy and Practice, 35(3) :225 – 241, 2001. (Cité en page 20.)

[51] Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. European Journal of Operational Research, 16(2) :236 – 245, 1984. (Cité en page 19.)

[52] E. F. Moore. The shortest path trough a maze. page 7, April 1957. (Cité en page 8.)

[53] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional a* search on time-dependent road networks. Networks, 59(2) :240–251, 2012. (Cité en page 15.)

[54] T. A. J. Nicholson. Finding the shortest route between two points in a network. The Computer Journal, 9(3) :275–280, November 1966. (Cité en page 9.)

[55] Andreas Paraskevopoulos and Christos Zaroliagis. Improved alternative route planning. In Daniele Frigioni and Sebastian Stiller, editors, ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems - 2013, volume 33 of OpenAccess Series in Informatics (OASIcs), pages 108–122, Sophia Antipolis, France, September 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cité en page 21.)

[56] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. J. Exp. Algorithmics, 12 :2.4 :1–2.4 :39, June 2008. (Cité en page 14.)

[57] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In Proceedings of the 13th annual European conference on Algorithms, ESA'05, pages 568–579, Berlin, Heidelberg, 2005. Springer-Verlag. (Cité en page 13.)

[58] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. In Proceedings of the 14th conference on Annual European Symposium - Volume 14, ESA'06, pages 804–816, London, UK, UK, 2006. Springer-Verlag. (Cité en page 13.)

[59] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. J. Exp. Algorithmics, 17(1) :1.6 :1.1–1.6 :1.40, September 2012. (Cité en page 13.)

[60] Gaël Sauvanet. Recherche de chemins multiobjectifs pour la conception et la réalisation d'une cen These, Université François Rabelais - Tours, April 2011. (Cité en page 20.)

[61] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line : an empirical case study from public railroad transport. J. Exp. Algorithmics, 5, December 2000. (Cité en page 13.)

[62] Dorothea Wagner and Thomas Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In Giuseppe Battista and Uri Zwick, editors, Algorithms - ESA 2003, volume 2832 of Lecture Notes in Computer Science, pages 776–787. Springer Berlin Heidelberg, 2003. (Cité en page 13.)

[63] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. Geometric containers for efficient shortest-path computation. J. Exp. Algorithmics, 10, December 2005. (Cité en page 13.)

[64] Jin Y. Yen. Finding the k shortest loopless paths in a network. Management Science, 17(11) :712–716, 1971. (Cité en pages 22 et 24.)

Annexe A

Réseau de transport et graphe

Un réseau de transport est l'ensemble des données liées à la représentation d'un ou de plusieurs modes de transport. La manipulation informatique d'un tel objet suppose de synthétiser ces informations en une structure de graphe comportant des propriétés sur ses arcs et ses noeuds.

Au vu des différences de nature des données et de leur déconcentration de part les multitudes de formats et de voies d'obtention, la compilation d'un graphe de transport constitue une tâche spécifique et complexe. D'autant plus que certaines informations peuvent être incomplètes voire erronées.
Lors de l'aggrégation des données, les sources indépendantes sont dénommées couches. La construction du graphe de transport consiste donc à aggréger l'ensemble des couches de transport en une seule entité. Parfois, il est plus adéquat de stocker les données non pas directement dans le graphe, mais de façon annexe en utilisant des identifiants permettant de rattacher ces données au graphe.

A.1 Données d'un réseau de transport

Voirie

La voirie, ou l'ensemble des voies de circulation, ou pour simplifier dans notre cas, l'ensemble des routes et des voies terrestres, constitue la base du réseau de transport. En effet, et de façon intuitive, c'est la couche la plus imposante quantitativement. Il s'agit également d'un espace incontournable qualitativement puisqu'il permet l'accès au bâti et à tout autre emplacement d'intérêt pour les utilisateurs. Cette couche de données sera donc utilisée comme référence pour toutes les autres. On peut également l'appeler réseau filaire, le tracé des routes et des chemins sans terre plein central étant représenté par un tronçon unique.

Les données de voirie peuvent être obtenues via différents fournisseurs. Les bases de données de *OpenStreetMap* proposent une cartographie mondiale qui est enrichie par des particuliers sur le même principe que Wikipédia. Ce sont des données libres, accessibles à tous. La couverture est de bonne qualité en France mais nécessite un lourd travail de correction car certains tracés ne sont pas connectés au reste de la voirie comme c'est souvent le cas des ronds-points ou d'îlots déconnectés de ses voies d'accès, ce qui

en prescrirait leurs utilisations. On pourra également citer, pour la France, les bases de données *bdtopo* et *bdadresse* de l'*IGN*. Ces bases sont de qualité supérieure et accessibles gratuitement pour les missions de service public, l'enseignement et la recherche. Il existe également des données payantes émanant de professionnels ou équipementiers de la route ou du transport routier.

Transports en commun

Les organisateurs de transport en commun fournissent de plus en plus un accès libre à leurs données. Elles décrivent les positions géographique des arrêts ainsi que les heures de passage correspondant à l'offre théorique ou temps réel du trajet linéraiee d'un véhicule par rapport à la voirie.
Dans le cas de lignes à horaires, surtout pour le bus, le trajet, c'est à dire la séquence des arrêts desservis s'appelle une mission. Pour une même ligne on pourra donc dire que la séquence d'arrêts diffère en fonction des heures de la journée, il peut donc y avoir plusieurs missions par jour. Une course correspond au trajet le long d'une mission avec des horaires de passage à chaque arrêt, c'est la fiche horaire disponible à l'affichage sur les abris. On peut noter que l'heure de départ est supérieure à l'heure d'arrivée et que cela peut être une inégalité stricte. Il y aura alors un temps d'attente à l'arrêt, afin de prendre en compte des ralentissements liés à des arrêts saturés et pour lesquels le temps de descente puis de montée des passagers est non négligeable.

Les formats libres couramment utilisés sont le *GTFS* (Google Transit Feed System [11]) dans le monde ou le format *Neptune* (*Norme d'Echange Profil Transport collectif utilisant la Normalisation Européenne* [8]) en Europe. Ces formats permettent de décrire de façon géographique l'offre de transport à savoir le positionnement des arrêts, les horaires de passage et parfois les tracés réels des lignes dans le cas du *GTFS*.

Il faut noter que le format *Neptune* peut prendre en compte le transport à la demande et les lignes à fréquences comme le métro. Lorsque cette modélisation n'est pas possible dans un format, les vitesses moyennes sont enregistrées comme des lignes à horaires, mais il s'agit là plus d'un problème de modélisation que de données. En effet, dans le cas d'un transport à vitesse moyenne, il y a toujours un transport arrivant après un temps d'attente égal à la fréquence et il est donc possible de reconstruire des horaires de passage à partir de la fréquence et de l'heure de début de service.

A.2 Transformation en graphe

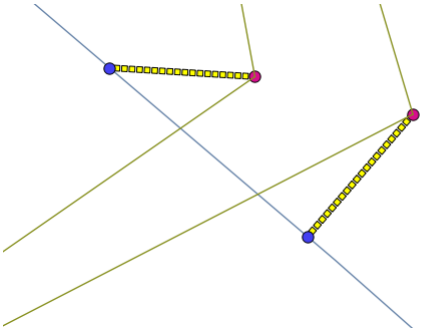
Pour obtenir un graphe à partir du réseau de transport, il faut relier les différentes couches entre elles en projetant les couches de transport en commun sur la voirie. Les points créés sont appelés nœuds de transferts, ils permettent de passer d'une couche à l'autre de façon transparente pour l'utilisateur du graphe. En chacun de ces points,

le filaire voirie est redécoupé pour créer un nœud du graphe et connecter les couches. Enfin, un arc de transfert relie les nœuds de transfert aux nœuds respectifs de leur couche d'origine, liant ainsi les deux couches.

Parfois, il peut être utile de connecter directement deux couches de transport en commun entre elles sans avoir à passer par la voirie. C'est notamment le cas lors de transferts souterrains entre différents transports, pour lesquels repasser par la rue n'a aucune validité pratique.

Le schéma de la figure A.1 illustre la projection de deux arrêts de transport en commun sur la voirie et l'ajout d'arcs de transfert. Les arrêts de transport en commun sont représentés par les nœuds en rouge et ils sont projetés sur la voirie via la création des nœuds en bleu. Les arcs de transfert (en trait épais) relient les arrêts et leur projection.

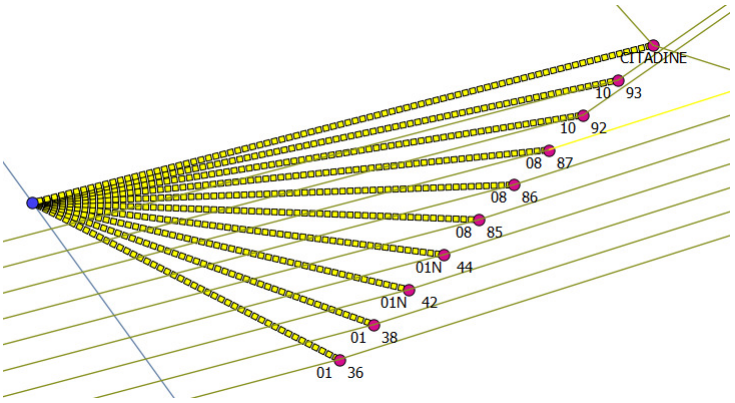
FIGURE A.1 – Arrêts projetés



Le schéma de la figure A.2 présente, pour un arrêt donné sur le réseau de transport en commun, la notion de lignes de transport et de mission associées. Pour chaque transport en commun présent à cet arrêt, on donne un identifiant de ligne (01, 01N, 08, 10, $CITADINE$ sur la figure) complété par un identifiant de mission. Pour la ligne 8 par exemple, il existe trois missions différentes passant par cet arrêt. Cet arrêt de transport en commun est dupliqué autant de fois qu'il y a de lignes et de missions (ensemble des nœuds en rouge de la figure) mais tous ces arrêts sont projetés sur le même nœud de voirie (nœud en bleu). On peut noter que pour les missions de cette figure, le tracé à vol d'oiseau ne change pas puisque les lignes sont parallèles, c'est à dire que les points précédants et suivants sont identiques.

Concernant les tables horaires, elles contiennent pour chaque arrêt, mission, course et horaire un temps de parcours. Une clé contenue dans les nœuds indexe la table permettant de retrouver pour un nœud tous les horaires des missions et courses passant par cet arrêt. Lors de la recherche d'un trajet, l'exploration du graphe fournit les numéros de l'arrêt et de la mission lorsqu'on monte dans un transport en commun. Il faut donc déterminer la première course passant après l'heure d'arrivée à l'arrêt courant. Une fois la course connue, dans les itérations suivantes, l'heure d'arrivée au prochain arrêt est obtenue immédiatement en suivant les arcs correspondant à la ligne et à la course, en récupérant les temps de trajets grâce à la mission.

FIGURE A.2 – Lignes et missions sur un arrêt



Lors de la compilation du réseau de transport en un graphe, le choix du numéro des sommets n'est pas sans importance même s'il peut être effectué a posteriori par isomorphisme. En effet, il est plus efficace de générer des nœuds dont l'identifiant est proche voire voisin de ses successeurs. Ainsi, lors du parcours du graphe, la localité spatiale des nœuds pourra accélérer l'exécution des algorithmes.

Annexe B

Langages

Un **alphabet** est un ensemble d'éléments appelés **lettres** dont la séquence forme un **mot**. L'opération associative interne sur les mots qui a pour zéro le mot vide est la **concatenation**.
L'ensemble de tous les mots sur un alphabet est appelé le **monoïde libre** engendré par l'alphabet. Un **langage** est une partie du monoïde libre.

Les langages ont été classifiés par Chomsky [29] en 1956 selon quatre hiérarchies :

- généraux, reconnaissablent par des machines de Turing ;
- contextuels, reconnuent par des machines de Turing linéairements bornées ;
- algébriques, reconnaissablent par des automates à pile ;
- rationels, reconnus par des automates.

B.0.1 Langage rationels

Le théorème de Kleene affirme que les langages rationels sont l'ensemble des langages reconnus par les automates finis.
Etant donné un certain nombre fini d'**états** E, Σ un alphabet et $\Delta : E \times \Sigma \to \mathbb{P}(E)$ une fonction de **transition**, un automate non-déterministe est le quintuplet $(E, \Sigma, I, F, \Delta)$ où $I \subseteq E$ est l'ensemble des états **initiaux** et $F \subseteq E$ l'ensemble des états **finaux**.

B.0.2 Langage algébrique

L'ensemble des langages algébrique sont reconnus par des automates à pile qui sont des automates auquels sont adjoints une zone de travail de type pile (premier entré - dernier sorti).