Calcul d'itinéraire multicritère en transport multimodal

Alexandre Iglesias

N°d'ordre NNT : 2017LYSEM025

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'École des Mines de Saint-Etienne

Ecole Doctorale N° 488
Sciences, Ingénierie, Santé

Spécialité de doctorat :
Discipline : Informatique

Soutenue publiquement le 12/10/2017, par :
Alexandre Stéphane Gonzalo IGLESIAS

Calcul d'itinéraire multicritère en transport multimodal

Devant le jury composé de :

Artigues, Christian - Directeur de Recherches - LAAS-CNRS	Président
Huguet, Marie-José - Professeur INSA-Toulouse - LAAS-CNRS	Rapporteur
Neron, Emmanuel - Professeur - École Polytechnique Université de Tours	Rapporteur
Wolfler Calvo, Roberto - Professeur - Université de Paris Nord	Examinateur
Feillet, Dominique - Professeur - Mines Saint-Etienne	Directeur
Quadri, Dominique - Maître de conférences HDR - Université Paris Sud	Co-directrice
Lesens, Franck- Directeur Technique - Cityway	Invité industriel

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	X. Delorme, Maître assistant
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'Etat ou d'une HDR)

ABSI	Nabil	CR	Génie industriel	CMP
AUGUSTO	Vincent	CR	Image, Vision, Signal	CIS
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BADEL	Pierre	MA(MDC)	Mécanique et ingénierie	CIS
BALBO	Flavien	PR2	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA(MDC)	Informatique	FAYOL
BLAYAC	Sylvain	MA(MDC)	Microélectronique	CMP
BOISSIER	Olivier	PR1	Informatique	FAYOL
BONNEFOY	Olivier	MA(MDC)	Génie des Procédés	SPIN
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BRUCHON	Julien	MA(MDC)	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	MA(MDC)	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	CR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA(MDC)	Génie industriel	Fayol
DELAFOSSE	David	PR0	Sciences et génie des matériaux	SMS
DELORME	Xavier	MA(MDC)	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR1	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
DOUCE	Sandrine	PR2	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
FAUCHEU	Jenny	MA(MDC)	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	CR	Génie des Procédés	SPIN
FEILLET	Dominique	PR1	Génie Industriel	CMP
FOREST	Valérie	MA(MDC)	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Sciences de la Terre	SPIN
GAVET	Yann	MA(MDC)	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GONDRAN	Natacha	MA(MDC)	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA(MDC)	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
GUY	Bernard	DR	Sciences de la Terre	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MOUTTE	Jacques	CR	Génie des Procédés	SPIN
NEUBERT	Gilles			FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
NORTIER	Patrice	PR1	Génie des Procédés	SPIN
O CONNOR	Rodney Philip	MA(MDC)	Microélectronique	CMP
OWENS	Rosin	MA(MDC)	Microélectronique	CMP
PERES	Véronique	MR	Génie des Procédés	SPIN
PICARD	Gauthier	MA(MDC)	Informatique	FAYOL
PIJOLAT	Christophe	PR0	Génie des Procédés	SPIN
PINOLI	Jean Charles	PR0	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	MR	Génie des Procédés	CIS
ROUSSY	Agnès	MA(MDC)	Microélectronique	CMP
ROUSTANT	Olivier	MA(MDC)	Mathématiques appliquées	FAYOL
SANAUR	Sébastien	MA(MDC)	Microélectronique	CMP
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia	Ingénieur de recherche	Microélectronique	CMP
VALDIVIESO	François	PR2	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR1	Génie industriel	CIS
YUGMA	Gallian	CR	Génie industriel	CMP

Mise à jour : 03/02/2017

À Capucine.

Remerciements

Cette thèse a été réalisée dans le cadre d'une convention industrielle entre Cityway et l'École Nationale des Mines de Saint-Étienne. Je remercie donc l'ensemble des personnes de ces organismes qui ont rendu possible ces travaux.

Je remercie en particulier mon directeur de thèse Dominique Feillet, directeur du laboratoire SFL au Centre Georges Charpak de Gardanne, qui a su se rendre disponible malgré ses responsabilités administratives. Travailler avec lui fut un véritable plaisir, et il m'a orienté avec pertinence et beaucoup de compétence tout au long de mes travaux de recherche. Mes remerciements vont également à Dominique Quadri, maître de conférences HDR à Université Paris Sud, co-directrice de ma thèse, pour sa gentillesse et sa patience lors des réunions au laboratoire ou à distance et pour ses précieuses relectures du manuscrit. Merci aussi aux rapporteurs de thèse et au jury de ma soutenance, Marie-José Huguet, Emmanuel Néron, Christian Artigues et Roberto Wolfer Calvo pour les remarques pertinentes et les critiques positives dont ils m'ont fait l'honneur, que ce soit dans leurs rapports écrits ou pendant la soutenance.

Je remercie également Franck Lesens et Laurent Briant, respectivement directeur technique et directeur général de Cityway, à l'initiative de ces longs travaux de recherche, et qui les ont accompagné lors des réunions de suivi, jusqu'à la soutenance. Je remercie chaleureusement Charlie Jacquelin, développeur Cityway sur le calculateur d'itinéraire, qui a repris une grande partie de mes tâches lorsque le temps pour terminer la thèse s'est fait court, et avec qui les conversations autour de ce sujet furent enrichissantes. Merci encore à Pascal Lechalier, responsable de l'équipe Data à Cityway, pour sa bonne humeur constante et son écoute attentive des problèmes que je lui ai exposés.

Je remercie évidemment les permanents et doctorants du laboratoire SFL pour leur accueil chaleureux, avec une mention spéciale pour les organisateurs des séminaires annuels, toujours réussis.

Je remercie Aurélia Lercari, ma fiancée, pour son soutien indéfectible et toutes ses attentions au quotidien qui m'ont donné le courage de continuer.

Pour terminer, je remercie ma famille et mes amis de m'avoir soutenu et encouragé dans cette voie, en particulier mes parents, sans lesquels je ne me serais pas lancé dans cette aventure, mon père pour m'avoir aidé à formaliser un certain nombre de notions, et Nicolas Wodniack pour son dévouement et sa disponibilité lors de la finalisation des trois premiers chapitres.

Table des matières

Introduction

Les réseaux de transport en commun sont de plus en plus complexes et denses, en particulier dans les grandes métropoles. Par conséquent, l'offre de transport nécessite des outils d'information et d'aide à la décision à la hauteur de cette complexité. Alors que la fiche horaire et les plans de lignes en sont les plus simples exemples, le **calculateur d'itinéraire** s'impose comme l'outil de référence, simplifiant grandement pour l'usager la planification d'un trajet dans une agglomération étendue, voire une région entière ou un pays.

Il est crucial que les trajets obtenus avec cet outil soient cohérents avec l'offre, et proposent à l'utilisateur des choix pertinents selon plusieurs critères, à commencer par l'horaire d'arrivée au plus tôt. De plus, chaque utilisateur ayant une vision subjective de ce que pourra être le meilleur trajet, il n'est pas évident qu'un programme informatique ait toute l'intelligence nécessaire pour proposer des résultats de qualité. Les algorithmes utilisés sont généralement des algorithmes de recherche de **plus court chemin** dans un graphe. Cependant un certain nombre de contraintes, comme la dépendance des temps de trajets en fonction de l'horaire et la nature multimodale des trajets, nécessitent l'établissement d'algorithmes et de règles complexes. Les problèmes que nous cherchons à résoudre varient selon plusieurs paramètres et scénarios, et les algorithmes développés doivent être efficaces dans tous les cas. Enfin, il est attendu que ces algorithmes de recherche d'itinéraire s'effectuent en des temps très brefs, inférieurs à la seconde, car leur finalité est de renseigner rapidement un utilisateur sur une page Web ou une application mobile.

La **multimodalité** et l'intermodalité se définissent comme la possibilité d'emprunter plusieurs modes de transport différents, exclusivement ou en combinaison, pour aller d'un endroit à l'autre. Un calculateur d'itinéraire doit intégrer cette fonctionnalité, qui vient s'ajouter aux scénarios envisagés. En effet, la recherche d'itinéraire prend en compte des étapes de transports en commun, vélo, voiture, marche à pied, etc. Nous avons donc énormément de possibilités de trajets différents.

En résumé, les principales difficultés de ce type de programme sont les suivantes :

- Modéliser, sous forme de graphe et en tant que données informatiques, toute la **complexité des réseaux de transport**. Il est également difficile de modéliser le fait qu'en transport en commun, la faisabilité et la durée d'un trajet dépendent de l'horaire.
- Mettre en œuvre les différentes **contraintes** : prise en compte des conditions de circulation, des restrictions de manœuvres, des temps minimum de changements à

l'arrêt, des interdictions de montée ou descente à certains arrêts, des interdictions de modes spécifiques, etc.

- Assurer des **temps de calcul** praticables : bien qu'une application de l'algorithme de Dijkstra soit rapide pour des petits réseaux, la multimodalité et la prise en compte des transports des grandes métropoles peuvent le ralentir considérablement. De plus, il est fréquent de vouloir implémenter une *range query*, c'est-à-dire une requête dont la contrainte de départ est une plage horaire, plutôt qu'un horaire précis.
- Assurer la **qualité** des solutions obtenues : il faut que les utilisateurs du calculateur d'itinéraire soient satisfaits des résultats. En effet, trouver un trajet optimisant une agrégation de critères (somme pondérée de temps de trajet, nombre de changements, distance de marche à pied...) ne suffit pas à rendre compte de la diversité des solutions préférables et préférées par l'utilisateur. La prise en compte de **plusieurs critères** simultanément transforme le problème polynomial en problème NP-complet (en général).

La littérature spécialisée propose un certain nombre de solutions à ces problèmes. Sur les questions de modélisation, il est naturel de représenter un réseau de transport par un ou plusieurs graphes. En général, un nœud représente un lieu géographique, et un arc un déplacement. La dépendance au temps se modélise de deux façons : soit par un changement de modèle (chaque nœud représente un évènement), soit par l'ajout d'une fonction d'évaluation de chaque arc en fonction du temps. L'exploitation de ce graphe se fait principalement par un algorithme de parcours des arcs basé sur une file de priorité. Selon la nature du graphe, elle peut aussi se faire par d'autres méthodes, comme le tri topologique par exemple. Sur un autre plan, la résolution des difficultés liées à la normalisation, l'intégration et la vérification des données cartographiques et de transport en commun, bien qu'intéressante et importante, n'est pas développée dans nos travaux.

Concernant la prise en compte de contraintes, chacune est d'une nature spécifique, et s'implémente par conséquent avec une méthode spécifique. Par exemple, les restrictions de manœuvres sont traitées par l'utilisation du graphe adjoint, ou par la mise en place de règles de calcul supplémentaires. La prise en compte des temps de transfert se fait en modifiant l'évaluation des arcs de montée et/ou de descente des transports à l'arrêt.

Concernant les temps de calcul, de nombreux travaux accélèrent grandement ces derniers sur plusieurs types de graphes, moyennant des temps de prétraitement plus ou moins lourds. Notons que toutes les méthodes d'accélération ne s'appliquent pas à tous les problèmes. En particulier, les méthodes s'appliquant sur les graphes de réseaux routiers sont extrêmement efficaces, avec des temps de calcul moyens d'une requête simple de l'ordre de la microseconde, dans un réseau comme l'Europe de l'Ouest. En revanche, ces méthodes sont en général trop rigides pour s'appliquer à des cas complexes, en particulier lorsque nous traitons un graphe dépendant du temps, avec beaucoup de contraintes et plusieurs critères de recherche.

Concernant la qualité des solutions, il faut nécessairement générer plusieurs solutions différentes pour la même requête avec un calcul multicritère, et en sélectionner les meilleures. La difficulté liée au calcul de plus court chemin multicritère se contourne principalement en raison des arguments suivants : en premier lieu, les critères sont

souvent positivement corrélés, diminuant l'explosion du nombre de chemins optimaux dans le graphe. De plus, des heuristiques de dominance entre les trajets sont utilisées pour réduire encore ce nombre. Enfin, il existe des méthodes de sélection de solutions *a posteriori* en fonction de plusieurs critères, comme la méthode des k-moyennes.

C'est la combinaison de ces difficultés et méthodes de résolution, ainsi que la taille croissante des instances sur lesquelles nous souhaitons calculer des itinéraires, qui constitue l'obstacle majeur des divers travaux sur le sujet, y compris du nôtre.

Présentation de Cityway

Cette thèse a été effectuée dans le cadre d'un partenariat industriel avec la société **Cityway**. Cityway est une société de services créée en 2001 à l'initiative de Veolia Transport, devenue depuis Transdev, dans le but de fournir des informations appliquées au domaine de la mobilité : sites internet, applications mobiles, calculateurs d'itinéraire, suivi des véhicules, systèmes billettiques, centres de relation clients et gestion de pôle d'échanges. Elle travaille pour les réseaux de transport, les collectivités locales et les grands acteurs publics. Elle est également réputée pour sa contribution aux démarches de normalisation et de partage de données transport à l'échelle européenne.

Son calculateur d'itinéraire intègre tous les modes de transports disponibles, en particulier les transports publics, y compris le transport à la demande, la marche à pied, le vélo personnel mais aussi le vélo en libre-service, la voiture particulière, en covoiturage, et en auto-partage.

Régulièrement, de nouveaux clients et de nouvelles demandes poussent Cityway à l'innovation sur le calculateur d'itinéraire, que ce soit par des contraintes supplémentaires, l'intégration de règles ou de données d'un nouveau type, ou un fonctionnement innovant, comme la prise en compte des horaires temps réel ou de l'accessibilité des moyens de transport.

Objectif de la thèse

Nous nous intéressons à l'amélioration du calculateur d'itinéraire de Cityway. Ce calculateur, écrit en langage C avant la création de la société, implémente une version spécifique au transport en commun d'un algorithme monocritère de plus court chemin dans un graphe. Écrit en plusieurs dizaine de milliers de lignes de code mélangeant les langages C et C++, il assure un grand nombre de types de requêtes, avec de nombreux paramètres, et est extrêmement configurable. Il a été amélioré pendant plus de quinze années par plusieurs développeurs successifs.

Nous nous sommes fixés comme objectif de prendre du recul sur le code, et, en s'appuyant sur la littérature relative au sujet, de chercher des voies d'amélioration pérennes du produit, tout en conservant ses particularités et son intégration dans la chaîne de produits Cityway.

En particulier, nous avons essayé de résoudre les problèmes que présentent l'algorithme de Dijkstra minimisant une somme pondérée de divers sous-critères. Nous avons

donc réfléchi à ce que pourrait apporter l'implémentation d'algorithmes multicritères, et ce que cela pourrait coûter en termes de performances.

Enfin, nous nous sommes intéressés à la diversité des solutions, en remarquant que bien souvent, une solution peut être sous-optimale mais intéressante à présenter à l'utilisateur final pour des raisons variées dont nous n'avons pas forcément la maîtrise.

Principales contributions

Notre première contribution a été d'établir un état de l'art le plus complet possible, et de faire une analyse comparative de l'existant académique et industriel avec le produit de Cityway.

Dans un second temps, nous avons amélioré l'algorithme du calculateur de Cityway pour en faire un véritable algorithme multicritère multilabel, tout en faisant en sorte qu'il n'y ait pas de régression en termes de fonctionnalités et de configurabilité. Nous avons effectué des campagnes de tests de cet algorithme bicritère « arrivée au plus tôt, moins de changements » sur les données réelles de trois réseaux très différents. Nous avons explicité le critère implicitement utilisé par Cityway, qui est en réalité le critère lexicographique d'« arrivée au plus tôt puis départ au plus tard », et vu comment le traiter sans trop perdre en performance grâce aux propriétés de ce critère. Nous avons effectué d'autres campagnes de tests pour établir la pertinence de ces améliorations.

Enfin nous avons défini ce que nous entendons par diversité, et détaillé deux algorithmes pour trouver des solutions optimales mais de surcroît diverses à une recherche d'itinéraire. Le premier utilise une succession d'appels monolabels, dans un ordre déterminé par un découpage de l'ensemble des solutions réalisables. Le second utilise le nouvel algorithme multilabel. Dans les deux cas, nous avons complété l'algorithme par une méthode de sélection des solutions basée sur l'algorithme des k-means, et proposé des alternatives possibles à cette méthode.

Plan détaillé

Chapitre un : problème et modélisation. Ce chapitre introduit le vocabulaire de la recherche d'itinéraire sur des réseaux routiers et de transport en commun. Nous établissons formellement les définitions nécessaires à la modélisation du problème. Nous introduisons également les divers modèles utilisés dans la littérature pour le résoudre.

Chapitre deux : état de l'art sur les algorithmes de résolutions. Ce chapitre traite des algorithmes de plus court chemin, en particulier de l'algorithme fondateur de Dijkstra, ainsi que de certaines de ses nombreuses variantes et améliorations. Nous présentons également des algorithmes spécifiques à la recherche d'itinéraire dans un réseau de transport en commun.

Chapitre trois : recherche d'itinéraire chez Cityway. L'objectif de ce chapitre est de positionner le calculateur d'itinéraire de Cityway au sein de l'état de l'art. Après avoir détaillé son fonctionnement pour les réseaux de transport en commun et individuels, nous exposons l'algorithme distribué permettant de faire des calculs multimodaux. Enfin, une brève discussion sur les points faibles du calculateur permettent de justifier les axes d'évolutions proposés dans nos travaux.

Chapitre quatre : algorithme multilabel pour une optimisation multicritère et application à un cas bicritère. Ce chapitre introduit les concepts de base de la théorie multicritère, et détailles les adaptations nécessaires pour passer de l'algorithme de Dijkstra à l'algorithme dédié au cas multicritère. Nous présentons également un certain nombre de tests sur trois réseaux de transport très différents les uns des autres.

Chapitre cinq : génération et sélection d'un ensemble diversifié de trajets alternatifs. Nous explicitons dans ce chapitre ce que nous entendons par diversité des solutions. Pour cela, nous définissons une distance entre les trajets en transport en commun qui mesure à quel point ils sont ou ne sont pas similaires. Nous proposons deux algorithmes: l'un en monolabel, l'autre en multilabel pour obtenir un grand nombre de trajets réalisables presque optimaux selon le critère de l'arrivée au plus tôt ou le critère de somme pondérée. Enfin nous déterminons comment sélectionner des trajets pertinents à présenter à l'utilisateur parmi ces trajets réalisables.

Chapitre six : critère lexicographique de durée minimale. Nous nous attachons dans ce chapitre à mettre en exergue et justifier que le critère de l'arrivée au plus tôt ne suffit pas (il est également nécessaire de sélectionner la solution partant au plus tard) et nous montrons comment adapter cette problématique au cas multicritère. Nous démontrons et utilisons certaines propriétés mathématiques pour optimiser l'algorithme bicritère avec ce nouveau critère, et présentons des résultats expérimentaux sur nos jeux de tests.

Conclusion et perspectives. Ce chapitre final est consacré à la synthèse de nos résultats, ainsi qu'à leur utilisation potentielle pour améliorer le calculateur d'itinéraire de Cityway. Nous établissons également comment utiliser directement la nouvelle version du calculateur, ou s'en inspirer, pour résoudre un certain nombre de problèmes posés par les clients. Par exemple, intégrer un critère du moins cher constitue nos perspectives à court terme, et intégrer celui du trajet le plus robuste pourrait être une perspective à plus long terme.

Chapitre I

Problème et modélisation

Sommaire

Ce travail de thèse fait appel à de nombreux concepts et un vocabulaire spécifiques au métier du transport, à la cartographie de voirie et au calcul d'itinéraire dans des réseaux de transport en commun et de voirie. Nous ferons parfois appel à l'abréviation TC pour signifier « transport en commun » dans le reste de ce document.

La compréhension de chaque concept présenté en section 1 permet de mieux appréhender le problème posé. Leur modélisation en objets mathématiques (section 2) et la modélisation des réseaux sous forme de graphe (section 3), permettront par la suite d'apporter des méthodes de résolution.

1 Concepts généraux

Cette liste a vocation de lexique et non de définition formelle mathématique. Ces concepts sont très étroitement liés aux données nécessaires à l'automatisation de la recherche d'itinéraire.

Mode de transport Un mode de déplacement correspond à une façon de se déplacer d'un endroit à l'autre. C'est un ensemble défini en extension, qui inclut, mais n'est pas limité à : *voiture, vélo, taxi, marche, train, car, bus, avion, métro, tram, covoiturage.* Certains modes sont catégorisés comme **individuels** (voiture, vélo, marche), d'autres sont des modes de **transports en commun** (métro, bus, train, avion), et d'autres, que l'on appelle parfois **modes doux** sont entre les deux (voiture/vélo libre service, covoiturage, taxi, transports à la demande). On peut également les distinguer par le fait que certains sont affectés par les conditions de trafic (voiture, taxi, bus, car), et d'autres non (marche, vélo, bus sur voie des bus...).

1.1 Cartographie

Lieu géographique Un lieu ou point géographique est simplement un point sur le globe terrestre, géolocalisé dans un système de coordonnées géographiques, comme le WGS84 [1] ou le Lambert [2]....

Zone géographique Une zone géographique est une surface sur le globe terrestre, géolocalisée par polygone dans un référentiel donné (WGS84, Lambert...).

Lieu public Un lieu public ou point d'intérêt est un lieu ou une zone géographique spécifique, doté d'un nom et d'un intérêt particulier. Exemples : *Mairie du 14ème, Hôpital Saint Jérôme, Supermarché Casino, Gare de Lyon....* Pour simplifier la modélisation, on rapporte un lieu public à un lieu géographique, même s'il correspond physiquement à une zone géographique. On remplace alors la zone par son centroïde.

Région, Département, Commune Ces trois concepts sont des découpages géographiques administratifs. En cartographie, ils correspondent à une surface définie par un polygone. La commune est définie en France par son code INSEE. Exemple : *Provence-Alpes-Côte d'Azur, Bouches du Rhône, Aix-en-Provence.*

Polyline Une polyline est un ensemble de points géographiques ordonnés et reliés par des segments successifs. En mathématiques, on appelle cela une *ligne brisée*, ou encore une *ligne polygonale.*

Tronçon de voirie Un tronçon de voirie est une *polyline* qui compose une voirie, qui n'a pas d'intersections avec d'autres voiries à part à ses extrémités. Il a un LIEU GÉOGRAPHIQUE de début et un LIEU GÉOGRAPHIQUE de fin.

Voirie Une voirie est une voie de circulation, c'est-à-dire un chemin géographique qui permet d'aller d'un point à un autre par la marche, le vélo, la voiture par exemple. C'est un ensemble de tronçons de voirie qui porte le plus souvent un nom. En France, quand elle a un nom, une voirie est définie de manière unique par son nom et sa commune. Exemple : *Rue de la paix, Aix-en-Provence*

Adresse Une adresse est un quadruplet (*Numéro, Voirie, Commune, Pays*) qui définit avec précision un lieu géographique non nommé. Exemple : *19 rue d'Entrecasteaux, Aix-en-Provence, France* définit un couple (latitude, longitude) précis sur le globe.

Piste et bande cyclable Une piste cyclable est une voie de circulation particulière, en cela qu'elle permet de se déplacer en vélo de manière sécurisée, en dehors de la circulation des voitures. Une bande cyclable est une piste cyclable incluse sur la chaussée d'une voirie.

1. https://fr.wikipedia.org/wiki/WGS_84
2. https://fr.wikipedia.org/wiki/Projection_conique_conforme_de_Lambert

Provenance des données

Ces informations sont fournies par des services de cartographie et géolocalisation publics ou privés, tels que *Navteq*[3], *IGN*[4], *OpenStreetMap*[5]. Il est important de noter que ces données peuvent être plus ou moins complètes, plus ou moins exactes et plus ou moins à jour en fonction du fournisseur et de l'ancienneté de la carte.

1.2 Transport

Arrêt physique (Arrêt/Poteau/Quai/*Stop*) Un arrêt physique est un LIEU GÉOGRAPHIQUE à partir duquel il est possible de monter dans un véhicule de transport en commun ou d'en descendre. Exemple : *Gare de Lyon Quai A1, Gare de Lyon Quai A2, Gare de Lyon (Bus 1, 6, 8), Gare de Lyon (Bus 15, 18), Gare de Lyon (Métro)...*

Note : la plupart du temps, il n'y a qu'un arrêt pour tous les quais d'une gare, la modélisation des quais et des distances entre les quais n'est pas prise en compte.

Arrêt logique (Pôle d'échange/Gare/Station) Ces concepts correspondent à un regroupement d'ARRÊTS PHYSIQUES ou de quai par rapprochement géographique ou nominal. Exemple : *Gare de Lyon* qui regroupe tous les arrêts proche de la gare, y compris les arrêts de bus/métro. Cela permet de ne pas faire apparaître tous les arrêts physiques d'un même lieu public, aux noms souvent identiques, dans les listes de sélection d'arrêt de départ ou d'arrivée d'un trajet par exemple.

Horaire Un horaire est une indication d'une heure de passage d'un transport à un ARRÊT PHYSIQUE, pour un jour donné connu. Il est souvent exprimé sous la forme (heure, minute, seconde) ou par un entier représentant le nombre de minutes depuis minuit. Pour plus de précision, en général pour le transport ferroviaire, il peut y avoir deux horaires de passage : un horaire d'arrivée à l'arrêt, et un horaire de départ de l'arrêt.

Course (Journey) Une course est la description du parcours d'un véhicule de transport en commun. C'est en pratique une succession de couples (ARRÊT PHYSIQUE/HORAIRE), dont les horaires sont croissants, accompagnée du MODE de transport correspondant. Chaque course peut avoir, en plus du mode, d'autres attributs précisant par exemple le niveau d'accessibilité du transport (chaise roulante, non voyant...), ou leur niveau de service (voiture-restaurant, train-couchette, vélo à bord...).

Correspondance marche à pied (*Footpath*) Lorsque deux arrêts physiques sont suffisamment proches l'un de l'autre, et qu'il est possible de marcher de l'un à l'autre, on dit qu'il y a une correspondance entre ces arrêts. Cela nous indique qu'il est autorisé d'effectuer un CHANGEMENT d'une course desservant un arrêt à une autre desservant l'autre arrêt, moyennant un temps de marche. Cela remplace effectivement un ensemble de tronçons de voirie que l'on pourrait emprunter

3. https://here.com/en/navteq
4. http://professionnels.ign.fr/route500
5. https://www.openstreetmap.org/

entre deux arrêts. Prédéfinir cet ensemble permet d'une part d'optimiser les temps de calculs, mais aussi de mieux contrôler les correspondances jugées acceptables. Exemple : *(Gare de Lyon SNCF, Gare de Lyon Metro, 3 minutes, 250 mètres)*. Les arrêts physiques d'un même arrêt logique sont souvent en correspondance, mais il peut également exister des correspondances entre arrêts physiques d'arrêts logiques distincts.

Itinéraire (Route) A chaque COURSE, il est possible d'associer un itinéraire : c'est simplement la succession des ARRÊTS PHYSIQUES qu'elle dessert. Plusieurs COURSES peuvent avoir le même ITINÉRAIRE du moment qu'elles desservent les mêmes ARRÊTS PHYSIQUES dans le même ordre. On peut alors classer toutes les courses d'un même itinéraire par ordre croissant de temps de départ au premier arrêt. En pratique, une course n'en dépasse jamais une autre sur le même itinéraire.

Jours de fonctionnement (*Traffic days*) Les jours de fonctionnement sont associés à certaines courses et définissent quels jours de l'année elles fonctionnent.

Les concepts suivants sont moins fondamentaux pour l'établissement d'une recherche d'itinéraire, mais aident à la compréhension générale de ce qu'est un réseau de transport en commun.

Table horaire (*Timetable*) La TABLE HORAIRE d'un ARRÊT PHYSIQUE est l'ensemble des HORAIRES de passage des transports en commun à cet arrêt pour un jour donné.

Ligne Une LIGNE est un regroupement d'itinéraires (et donc des courses correspondantes) sous un nom. Dans les cas les plus simples, il y a deux ITINÉRAIRES par LIGNE : un ITINÉRAIRE sens aller et un ITINÉRAIRE sens retour. Il y a beaucoup d'exceptions, certaines COURSES peuvent omettre certains ARRÊTS par exemple, ou changer leur ordre à certaines heures. Les cas les plus complexes sont dans le domaine ferroviaire, ou la même LIGNE peut regrouper des ITINÉRAIRES très différents les uns des autres (par exemple le RER C à Paris).

Réseau Un ensemble d'arrêts connectés entre eux par des LIGNES et des CORRESPONDANCES MARCHE À PIED forment un RÉSEAU de transport en commun, qu'il est possible de parcourir en empruntant ses transports. Les lignes sont regroupées en réseaux pour des raisons administratives et financières.

Exploitant L'exploitant est une société ou une administration (Autorité Organisatrice des Transports) qui met en place et maintient le réseau de transport en commun dans un périmètre administratif donné.

1.2.1 Exemple

Exposons ces concepts dans un exemple de réseau composé de deux lignes et illustré par la figure 1.

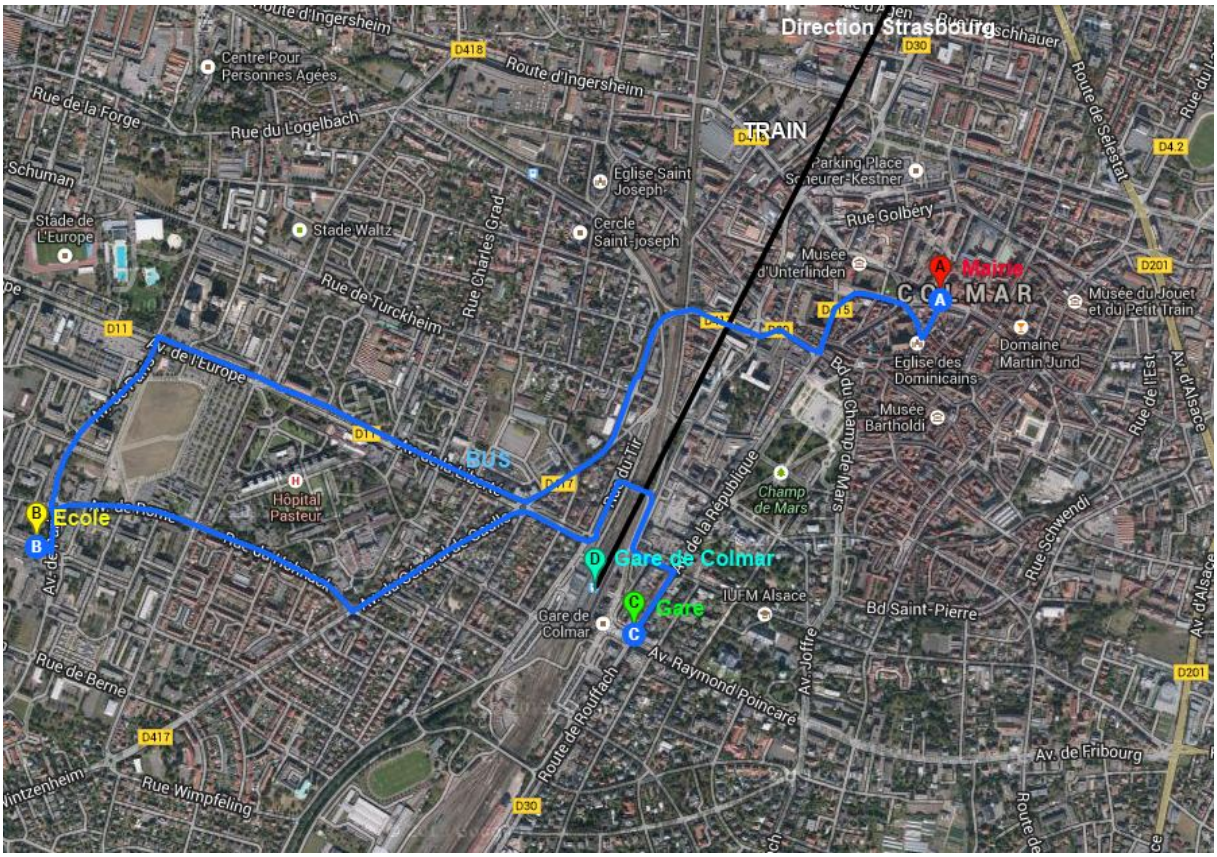


FIGURE 1 – Réseau TC fictif de la ville de Colmar

La ligne de bus *Mairie - Gare* passe par les trois arrêts suivants : Mairie, École, Gare. Le tableau 1 représente ses horaires de passage à ses arrêts. Elle est donc composée de trois itinéraires (deux allers et un retour) : I1 = (Mairie, École, Gare), I2 = (Gare, École, Mairie), I3 = (Mairie, Gare). Elle possède cinq courses : C1, C3 ont I1 pour itinéraire, C2 a I3 pour itinéraire et C4, C5 ont I2 pour itinéraire.

On peut imaginer que la course C2 (de 12h00) ne fonctionne que les weekends, ce qui induit les jours de fonctionnement suivants :

- C1, C3, C4, C5 : tous les jours de l'année sauf jours fériés et dimanche.
- C2 : tous les samedi et dimanche de l'année.

La ligne de train régional *Gare de Colmar - Gare de Strasbourg* relie quant à elle les villes Colmar et Strasbourg. Le tableau 2 représente ses horaires de passage à ses arrêts. Elle est composée deux itinéraires (un aller et un retour) : I4 = (Gare de Colmar, Gare de Strasbourg), I5 = (Gare de Strasbourg, Gare de Colmar). Elle possède quatre courses : C6, C7 ont I4 pour itinéraire et C8, C9 ont I5 pour itinéraire. Ces courses fonctionnent tous les jours de l'année.

Enfin, l'arrêt de train *Gare de Colmar* est suffisamment proche de l'arrêt de bus *Gare* pour justifier l'existence d'une correspondance marche à pied entre les deux.

Sens aller	C1	C2	C3
Mairie	8:00	12:00	18:00
École	8:15	|	18:15
Gare	8:25	12:18	18:25
Sens retour	C4	C5	
Gare	7:00	16:30	
École	7:05	16:34	
Mairie	7:25	16:50	

Tableau 1 – Horaires de la ligne *Mairie-Gare*

Sens aller	C6	C7
Gare de Colmar	7:50	15:30
Gare de Strasbourg	8:50	16:30
Sens retour	C8	C9
Gare de Strasbourg	09:30	17:35
Gare de Colmar	10:30	18:35

Tableau 2 – Horaires de la ligne *Gare de Colmar - Gare de Strasbourg*

1.2.2 Nomenclature

Dans la littérature, la nomenclature liée aux transports en commun varie légèrement d'un texte à l'autre, car elle dépend du milieu de transport auquel s'applique les travaux. Par exemple, dans [Schn2009] nous sont présentées les notions ferroviaires de Trains, Stations, Train classes, Attributes, Traffic days, Footpaths et Connections, qui correspondent respectivement aux notions de Course, Arrêt logique, Mode, attributs de Course, Jours de fonctionnement, Correspondance marche à pied, Trajet. Au final, quelle que soit la façon d'interpréter les données, on peut les ramener au lexique défini ci-dessus.

1.2.3 Normes

Un grand nombre de normes existent pour décrire précisément un réseau de transport en commun et peuvent généralement se rapporter aux concepts décrits ci-dessus. Par exemple, le tableau 3 décrit comment le format d'échange créé par Google, le GTFS (pour *General Transit Feed Specification*) s'intègre à notre lexique.

agency.txt	Exploitant
stops.txt	Arrêts physiques
routes.txt	Itinéraires
trips.txt	Courses
stop_times.txt	Table horaire
calendar.txt	Jours de fonctionnement

Tableau 3 – Fichiers Google Transit

1.3 Trajets

Trajet (Itinéraire/*Trip*/*Connection*) Un itinéraire, que l'on préférera appeler TRAJET pour ne pas le confondre avec l'ITINÉRAIRE d'une LIGNE, est la description détaillée et horodatée d'un parcours d'un lieu de départ à un horaire donné vers un lieu d'arrivée à un horaire d'arrivée.

Par exemple, un trajet possible dans le réseau donné en exemple est le suivant. Une première partie du trajet consiste à prendre le bus de 12h00 à l'arrêt Mairie pour arriver à l'arrêt Gare à 12h18. Une deuxième partie est une correspondance marche à pied de l'arrêt Gare jusqu'à l'arrêt Gare de Colmar. La dernière partie consiste à prendre le train de 17h35 à la gare de Colmar pour arriver à la gare de Strasbourg à 18h35.

Bien que nous ayons adopté le terme TRAJET au lieu d'itinéraire, nous continuons à parler de **calcul d'itinéraire** du fait de l'emploi courant de cette expression.

Étape (*Step*) Une ÉTAPE de trajet est une portion d'un trajet, délimitée par tout changement de MODE ou de COURSE, le cas échéant.

Le trajet ci-dessus comprend trois étapes : une de bus, une de marche à pied, et une de train.

Trajet monomodal Un TRAJET MONOMODAL est un trajet dont toutes les ÉTAPES partagent le même MODE, sans compter les étapes de marche à pied.

Montée, Descente (*Get in*, *Get off*, *Board*, *Alight*) Les termes MONTÉE et DESCENTE définissent respectivement les faits « d'embarquer dans » et de « débarquer d'un » véhicule de transport en commun. Cela correspond au début et à la fin d'une ÉTAPE de transport en commun.

Changement (Correspondance, Transfert) Un CHANGEMENT est défini au sein d'un trajet comme le passage d'une COURSE de transport en commun à une autre, en passant par une DESCENTE puis une MONTÉE, et éventuellement par une CORRESPONDANCE de marche à pied intermédiaire.

1.4 Définition du problème

Bien souvent, il existe de très nombreuses façons de se rendre d'un lieu géographique A à un lieu géographique B, de très nombreux TRAJETS sont possibles. Chacune de ses façons peut être plus ou moins longue, onéreuse, confortable, n'avoir qu'un mode ou en mélanger plusieurs, utiliser les transports en commun ou les modes individuels. Dans les grandes villes et autour, l'offre de transport en commun et le réseau routier sont si complexes qu'il est difficile pour un utilisateur, qu'il soit résident ou visiteur, de choisir un trajet pour ses déplacements quotidiens ou exceptionnels.

C'est dans l'optique de fournir à l'utilisateur un trajet susceptible de le satisfaire parmi tous les trajets possibles que le problème de la recherche d'itinéraire multimodale et multicritère se pose.

Une requête à un système d'aide à la recherche d'itinéraire est la formulation d'une question à laquelle la réponse est un ou plusieurs TRAJETS.

2 Formalisation

Maintenant que le vocabulaire concernant le transport multimodal et la recherche d'itinéraire a été défini, formalisons mathématiquement les concepts qui nous seront nécessaires par la suite.

2.1 Réseaux

Définition 1 *Un **mode** de transport est un élément de l'ensemble des modes* $\mathcal{M} = \{$*marche, vélo, voiture, bus, tram, car, metro, train...* $\}$*, défini par extension. C'est l'union des ensembles distincts :*

- *modes de transport en commun* $\mathcal{M}_{TC} = \{$*bus, tram, car, metro, train...*$\}$,
- *modes de transport individuel* $\mathcal{M}_i = \{$*voiture, vélo, marche à pied, taxi, roller...*$\}$ *et*
- *modes doux* $\mathcal{M}_d = \{$*vélo en libre service, autopartage, transport à la demande, covoiturage...*$\}$

2.1.1 Réseau de transport individuel

Définition 2 *Un **lieu géographique** ou plus simplement **lieu** est un point de la terre qui est représenté par des coordonnées* $(x, y) \in \mathbb{R}^2$ *auxquelles se rajoute parfois l'altitude* z*.*

C'est un espace métrique et on notera $\texttt{dist}(L_1, L_2)$ la distance euclidienne usuelle entre les lieux L_1 et L_2.

Définition 3 *Un* ***tronçon géographique*** *est un tuple de lieux géographiques* $t = (L_1, ..., L_n)$ *avec* $n > 1$*, définissant sa forme géographique. On note* $\mathit{ref}(t) = L_1$ *et* $\mathit{nref}(t) = L_n$ *ses extrémités.*

Définition 4 *Un* ***réseau de voirie*** *est formalisé par les données :*

- *d'un ensemble de modes individuels* $\mathcal{M}_i$
- *d'un ensemble* $\mathcal{G}$ *de tronçons géographiques*
- *des attributs relatifs à chaque tronçon* t *:*
  - ▷ *la longueur* $\mathit{length}(t) = \sum_{i=1}^{n-1} \mathit{dist}(L_i, L_{i+1})$
  - ▷ *la navigabilité pour un mode individuel donné* $m \in \mathcal{M}_i$ *:* $\mathit{allowed}(t, m) \in \{0, 1\}$
  - ▷ *la vitesse moyenne de parcours pour un mode individuel donné* $m \in \mathcal{M}_i$ *:* $\mathit{speed}(t, m) \in \mathbb{R}^+$,
  - ▷ *le sens de circulation possible pour un mode individuel donné* $m \in \mathcal{M}_i$*:* $\mathit{direction}(t, m) \in \{$*both, forward, backward*$\}$,
  - ▷ *un coefficient de sécurité pour un mode individuel donné* $m \in \mathcal{M}_i$ *:* $\mathit{securitycoef}(t, m) \in [0, 1]$.

À notre connaissance, le coefficient de sécurité n'est utilisé en pratique que dans le cas des modes vélos ou marche à pied. Ce coefficient ne sera pas utilisé dans la suite de nos travaux.

2.1.2 Réseau de transport en commun

Définition 5 *Un* ***horaire*** *est un élément d'un ensemble discret* $\Pi \subset \mathbb{N}$ *d'horaires représentant le temps à partir d'un temps de référence donné et dans une unité donnée.*

En TC, $\Pi = [0; 1439]$ l'ensemble des minutes d'une journée, ou parfois $\Pi = [0; 86399]$ l'ensemble des secondes. Pour améliorer le modèle, on peut permettre de passer à la journée suivante en intégrant plus d'une journée dans Π ($\Pi = [0; 1679]$, par exemple).

Définition 6 *Un* ***arrêt physique*** *ou simplement* ***arrêt*** $A \in \mathcal{A}$ *est un* ***lieu géographique*** *particulier.*

Définition 7 *Un* ***évènement*** *est un couple* (A, τ)*, que l'on peut écrire* $A@\tau$*, avec* A *un lieu géographique et* τ *un horaire.*

La projection d'un évènement dans l'espace est un lieu et dans le temps un horaire.

Définition 8 *Une* ***course*** $C \in \mathcal{C}$ *est la donnée de* $(m, n, (A_i@h_i)_{i\in[1,n]})$*, avec* $n \in \mathbb{N}$ *le nombre d'arrêts qu'elle dessert,* $m \in \mathcal{M}_{TC}$ *son mode,* $A_i \in \mathcal{A}$ *et* $h_i \in \Pi$ *ses arrêts avec leurs horaires de passage. Par définition, une course ne remonte pas le temps :* $\forall i < j \in [1, n], h_i \leq h_j$.

Afin de définir formellement l'itinéraire d'une course, définissons la relation d'équivalence «partage le même itinéraire» $\mathcal{R}$ sur $\mathcal{C}$. Deux courses sont équivalentes au sens de $\mathcal{R}$ si elles partagent le même mode, la même succession d'arrêts et si leur ordre de passage relatif à chaque arrêt ne varie pas.

$$\begin{aligned}\forall c(m, n, (A_i @ h_i)_{i\in[1,n]}), c'(m', n', (A'_i @ h'_i)_{i'\in[1,n']}) \in \mathcal{C},\\ (c\mathcal{R}c') \Leftrightarrow (&m = m',\\ &n = n',\\ &\forall i \in [1, n], A_i = A'_i,\\ &\forall i < j \in [1, n], h_i \leq h'_i \Leftrightarrow h_j \leq h'_j)\end{aligned}$$

Définition 9 *L'**itinéraire*** $I(c)$ *d'une course* c *est la classe d'équivalence de* c *par la relation* $\mathcal{R}$*. On note* $\mathcal{I}$ *l'ensemble des itinéraires de* $\mathcal{C}$*.*

Définition 10 *Une **correspondance** marche à pied est un triplet* (A, B, τ) *avec* $A, B \in \mathcal{A}$ *et* $\tau \in \mathbb{N}$ *signifiant qu'il est possible de passer du lieu* A *au lieu* B *en* τ *temps.*

En général, $(A, B, \tau) \in \mathcal{T} \iff (B, A, \tau) \in \mathcal{T}$.

Définition 11 *Un **réseau de transport en commun** peut donc être formalisé par les données de :*

- Π *l'ensemble des horaires considérés.*
- $\mathcal{M}_{TC}$ = {*bus, tram, car, metro, train...* } *l'ensemble des modes de transport en commun.*
- $\mathcal{A}$ *l'ensemble des arrêts physiques de transport en commun.*
- $\mathcal{C}$ *l'ensemble des courses de transport en commun.*
- $\mathcal{T}$ *l'ensemble des correspondances du réseau.*

On peut déduire $\mathcal{I}$ *l'ensemble des itinéraires de* $\mathcal{C}$*.*

Définition 12 *Un **réseau multimodal** est la donnée d'un réseau de transport en commun et d'un réseau de voirie d'une même zone géographique.*

2.2 Étapes élémentaires et trajets

La notion d'**étape élémentaire** revient dans beaucoup de travaux ([Pyrg2004], [Pyrg2008], [Geis2010], [Schn2009], [Schu2005]), et permet de définir un trajet de manière formelle. De plus, elle est utilisée par certains algorithmes, comme le « *Connection Scan Algorithm* », présenté en chapitre II.

2.2.1 Notion d'étape élémentaire en transport en commun

Définition 13 *Étant donné un réseau de transport en commun, une* ***étape élémentaire de course*** *est un couple d'évènements qui se suivent directement dans une course du réseau. c'est-à-dire* $e = (c, A_d@h_d, A_a@h_a)$ *tels que* $c = (m, n, (A_i@h_i)_{i\in[1,n]}) \in \mathcal{C}$ *est une course et* $\exists i \in [1, n-1]/A_d@h_d = A_i@h_i \wedge A_a@h_a = A_{i+1}@h_{i+1}$.

C'est la plus petite décomposition d'un trajet de transport en commun.

On note l'ensemble des étapes élémentaires de course d'un réseau $\mathcal{E}$.

On note $c(e)$ la course d'une étape élémentaire de course e.

Définition 14 *Une* ***étape élémentaire TC*** *est*

- *ou bien une étape élémentaire de course telle que définie ci-dessus,*
- *ou bien un couple d'évènements* $e_c = (A_d@h_d, A_a@h_a)$ *avec* $(A_d, A_a, h_a - h_d) \in \mathcal{T}$ *représentant le parcours d'une correspondance marche à pied.*

On note $A_d(e)$ l'arrêt de départ, $A_a(e)$ l'arrêt d'arrivée, $h_d(e)$ l'horaire de départ, $h_a(e)$ l'horaire d'arrivée de l'étape élémentaire TC e.

On définit la **durée** d'une étape élémentaire TC $e \in \mathcal{E}$ comme le nombre positif ou nul $d(e) = h_a(e) - h_d(e)$.

Nous pouvons alors écrire un **trajet TC** en transport en commun comme une liste d'étapes élémentaires $T = (e_1, ..., e_k)$.

Définition 15 *On dit qu'un trajet TC* $T = (e_1, ..., e_n)$ *est* ***réalisable*** *sur un réseau de transport en commun s'il respecte les règles suivantes :*

- $\forall i \in [1, n-1], A_a(e_i) = A_d(e_{i+1})$ *: l'arrêt d'arrivée d'une étape élémentaire est l'arrêt de départ de la suivante.*
- $\forall i \in [1, n-1], h_a(e_i) \leq h_d(e_{i+1})$ *: l'horaire d'arrivée d'une étape élémentaire est inférieur ou égal à l'horaire de départ de la suivante.*

Notons que dans [Pyrg2008] et beaucoup d'autres travaux, les notions d'arrêts physiques et logiques sont confondues avec la notion de Station (gare) : les quais ne sont pas modélisés comme arrêts physiques et le passage d'un quai à l'autre ne constitue pas une étape élémentaire. Bien souvent, la durée d'un changement doit être supérieure à une donnée dépendant de la gare S appelée *transfer*(S), ou considérée nulle dans le cas simplifié.

2.2.2 Courses et itinéraires d'un trajet TC réalisable

Définition 16 *On appelle* ***courses*** *d'un trajet TC réalisable* $T = (e_1, \ldots, e_n)$ *la suite des courses* $(c_1, \ldots, c_k)$ *empruntées par ce trajet. Formellement, chaque étape*

élémentaire du trajet est soit une correspondance, soit une partie de course. Pour toute étape élémentaire e*, on écrit* $c(e)$ *la course de* e *si elle existe,* $\emptyset$ *sinon. Soit* $C'(T) = (c(e_1), \ldots, c(e_n))$*. On définit les courses d'un trajet* $\mathcal{C}(T)$ *comme la suite extraite de* $C'(T)$ *qui contient toutes les courses présentes dans* C'*, sans doublon et sans* $\emptyset$*.*

Pour un trajet donné, nous généralisons les notations A_d, A_a, h_d, h_a aux courses de ce trajet, même si une course est définie en dehors de l'existence d'un trajet.

- $A_d(c)$ l'arrêt de départ de la première étape élémentaire e du trajet telle que $c(e) = c$,
- $A_a(c)$ l'arrêt d'arrivée de la dernière étape élémentaire e du trajet telle que $c(e) = c$,
- $h_d(c)$ l'horaire de départ de la première étape élémentaire e du trajet telle que $c(e) = c$,
- $h_a(c)$ l'horaire d'arrivée de la dernière étape élémentaire e du trajet telle que $c(e) = c$,

Définition 17 *On appelle* ***itinéraire*** *d'un trajet réalisable* $T = (e_1, \ldots, e_n)$ *la suite des itinéraires* $(I_1, \ldots, I_k) = (I(c_1), \ldots, I(c_k))$ *empruntés par ce trajet,* $(c_1, \ldots, c_k)$ *étant les courses du trajet.*

2.2.3 Notion d'étape élémentaire et trajet en transport individuel

Nous allons étendre cette notion d'étape élémentaire au trajet individuel.

Définition 18 *Étant donné un réseau de voirie, une* ***étape élémentaire sur voirie*** *est un couple d'évènements représentant le parcours d'un tronçon de voirie dans un mode individuel.*

$e = (m, L_d@h_d, L_a@h_a)$ *est une étape élémentaire sur voirie si :*

- $m \in \mathcal{M}_i$ *i.e.* m *est un mode individuel,*
- *il existe un tronçon géographique* $g \in \mathcal{G}$ *du réseau de voirie tel que :*
- $\mathit{allowed}(g, m) = 1$,
- $h_a - h_d = \mathit{length}(g)/\mathit{speed}(g, m)$,
- *il est autorisé de parcourir* g *dans ce sens, i.e. l'une des conditions suivantes est remplie :*
  - ▷ $L_d = \mathit{ref}(g), L_a = \mathit{nref}(g), \mathit{direction}(g, m) \in \{\mathit{both}, \mathit{forward}\}$
  - ▷ $L_d = \mathit{nref}(g), L_a = \mathit{ref}(g), \mathit{direction}(g, m) \in \{\mathit{both}, \mathit{backward}\}$

Définition 19 *On dit qu'un trajet* $T = (e_1, ..., e_n)$ *sur voirie est* ***réalisable*** *s'il respecte les règles suivantes :*

- $\forall i \in [1, n-1], L_a(e_i) = L_d(e_{i+1})$ *: le lieu d'arrivée d'une étape élémentaire du trajet est le lieu de départ de la suivante.*
- $\forall i \in [1, n-1], h_a(e_i) \leq h_d(e_{i+1})$ *: l'horaire d'arrivée d'une étape élémentaire du trajet est inférieur ou égal à l'horaire de départ de la suivante.*

2.2.4 Trajet multimodal

Pour qu'un trajet multimodal soit possible, il faut que le réseau de voirie couvre les arrêts du réseau de transport. c'est-à-dire que les arrêts doivent être connectés au réseau de voirie par des tronçons géographiques, quitte à en rajouter : c'est le processus de *snapping*, décrit en annexe A de la thèse.

Définition 20 *Un **trajet réalisable** sur un réseau multimodal donné est une succession d'étapes élémentaires TC ou voirie telle que :*

- $\forall i \in [1, n-1], L_a(e_i) = L_d(e_{i+1})$ *: le lieu ou l'arrêt d'arrivée d'une étape élémentaire du trajet est le lieu ou l'arrêt de départ de la suivante.*
- $\forall i \in [1, n-1], \tau_a(e_i) \leq \tau_d(e_{i+1})$ *: l'horaire d'arrivée d'une étape élémentaire du trajet est inférieur ou égal à l'horaire de départ de la suivante.*

Nous verrons en section 3.3 les modèles permettant de composer les réseaux, et en section 3 du chapitre III les algorithmes utilisés par Cityway pour composer les trajets.

2.3 Recherche d'itinéraire

Étant donné un réseau R (TC, voirie ou multimodal), une **recherche d'itinéraire**, que l'on notera RI dans la suite de ce document, est un problème d'optimisation. On cherche le ou les trajets de R qui optimisent un ou plusieurs critères dans un sous-ensemble de trajets réalisables de R restreint par des contraintes.

Définition 21 *Une **contrainte** d'une RI est une proposition que doit respecter le trajet réalisable et qui définit l'ensemble des **trajets admissibles** de cette RI.*

Définition 22 *Un **critère** d'une RI est une fonction de Trajets(R) vers $\mathbb{R}$ que l'on cherche à minimiser ou maximiser.*

2.3.1 Contraintes usuelles

Ces définitions peuvent s'appliquer dans un contexte purement TC, dans un contexte voirie, ou dans un contexte multimodal et concerne les différents types de contraintes classiques de RI.

On appelle « premier évènement d'un trajet » le premier évènement de la première étape élémentaire du trajet.

On appelle « dernier évènement d'un trajet » le second évènement de la dernière étape élémentaire du trajet.

Définition 23 *Une* ***requête en départ à*** *ou* ***departure time query*** *est une RI dont la contrainte principale est la suivante :*

- *pour un* LIEU GÉOGRAPHIQUE *de départ* A*,*
- *un* LIEU GÉOGRAPHIQUE *d'arrivée* B*,*
- *un* HORAIRE τ *de départ minimal,*

le premier évènement s'écrit $\mathsf{A}@\tau_\mathsf{A}$ *avec* $\tau_\mathsf{A} \geq \tau$ *et le dernier évènement s'écrit* $\mathsf{B}@\tau_\mathsf{B}$ *avec* τ_B *quelconque.*

Dans [Geis2010], cette contrainte s'écrit $\mathsf{A}@\tau \rightarrow \mathsf{B}$, que l'on comprend comme : en partant de A à l'heure τ, pour aller vers B.

Définition 24 *Une* ***requête en arrivée à*** *ou* ***arrival time query*** *est une RI dont la contrainte principale s'écrit* $\mathsf{A} \rightarrow \mathsf{B}@\tau$ *et signifie que le dernier évènement s'écrit* $\mathsf{B}@\tau_\mathsf{B}$ *avec* $\tau_\mathsf{B} \leq \tau$ *et le premier évènement s'écrit* $\mathsf{A}@\tau_\mathsf{A}$ *avec* τ_A *quelconque.*

Définition 25 *Une* ***requête en fourchette de départ à ou range query*** *est une RI dont la contrainte principale est la suivante :*

- *pour un* LIEU GÉOGRAPHIQUE *de départ* A*,*
- *un* LIEU GÉOGRAPHIQUE *d'arrivée* B*,*
- *une plage d'*HORAIRE $[\tau_{\min}, \tau_{\max}]$ *de départ,*

le premier évènement s'écrit $\mathsf{A}@\tau_\mathsf{A}$ *avec* $\tau_\mathsf{A} \in [\tau_{\min}, \tau_{\max}]$ *et le dernier évènement s'écrit* $\mathsf{B}@\tau_\mathsf{B}$ *avec* τ_B *quelconque.*

2.3.2 Critères usuels

Définition 26 *Le critère d'****arrivée au plus tôt*** *cherche à minimiser l'horaire d'arrivée* τ_B *au lieu* B *en priorité. On appelle ce problème le « Earliest Arrival Problem » (EAP).*

Le critère d'**arrivée au plus tôt** ne s'applique jamais à une requête en *arrivée à*, puisque le résultat n'est pas intéressant : c'est simplement le trajet qui arrive le plus tôt dans Π.

Définition 27 *Le critère de* ***départ au plus tard*** *est la maximisation de l'horaire de départ* τ_A *au lieu* A *en priorité.*

Le critère de **départ au plus tard** ne s'applique jamais à une requête en *départ à*, puisque le résultat n'est pas intéressant : c'est simplement le trajet qui part le plus tard dans Π.

Définition 28 *Le critère* ***lexicographique de durée minimale*** *est un critère lexicographique composé de deux critères : le critère d'****arrivée au plus tôt*** $\mathtt{f}_1$ *et le critère de* ***départ au plus tard*** $\mathtt{f}_2$*. Si le problème est en départ à, le* ***critère lexicographique de durée minimale*** *est défini comme le critère lexicographique canonique composé de* $\mathtt{f}_1$ *puis* $\mathtt{f}_2$*. Si le problème est en arrivée à, le* ***critère lexicographique de durée minimale*** *est défini comme le critère lexicographique canonique composé de* $\mathtt{f}_2$ *puis* $\mathtt{f}_1$*.*

Dans la suite de notre document, il sera parfois fait mention de ce critère simplement comme du critère **lexicographique**.

Comme son nom l'indique, ce critère discrimine les trajets en fonction de leur durée parmi les trajets égaux sur leurs horaires d'arrivée (en *départ à*) ou de départ (en *arrivée à*).

C'est une variation indispensable du problème de recherche d'itinéraire dans un réseau de transport en commun, à laquelle le chapitre VI du présent document est consacré.

Définition 29 *Le critère du* ***moins de changements*** *est la minimisation du nombre de courses empruntées.*

Ce critère est nécessairement associé à un autre critère de type **arrivée au plus tôt** ou **départ au plus tard**.

2.3.3 Multicritère et critères additionnels

La liste des critères est vaste, comme le montrent les exemples ci-dessous :

- Arriver le plus tôt possible (EAP : Earliest Arrival Problem).
- Minimiser la durée du trajet.
- Minimiser le nombre changement (MNTP : Minimum Number of Transfers Problem).
- Minimiser le prix ([Schn2009]).
- Minimiser la durée de marche à pied.
- Maximiser le confort ([Schn2009]).
- Maximiser le temps pour dormir dans les trains de nuit ([Schn2009]).
- Minimiser le risque de manquer une correspondance (marge de temps dans [Schn2009], stochastique dans [Stra2012]).
- Minimiser l'émission de CO_2.
- Maximiser la proportion de pistes cyclables en vélo.
- Minimiser les montées en vélo.
- Minimiser la proportion de routes très sinueuses.

Dans la pratique, optimiser un seul de ces critères est assez limité : un trajet comportant six changements mais plus rapide d'une minute qu'un trajet direct n'intéressera dans la pratique personne. De même, un trajet direct qui part à 8h du matin mais fait arriver à 18h est beaucoup moins pertinent qu'un trajet partant à 8h, arrivant à 9h et comportant un changement par exemple.

Ainsi est introduit le problème multicritère de recherche d'itinéraire. On ne cherche non plus la solution optimale sur un seul critère, mais soit un compromis sur plusieurs critères, soit un ensemble de solutions non dominées sur chaque critère, dans lequel l'utilisateur ou un algorithme pourra choisir des solutions pertinentes.

2.3.4 Contraintes additionnelles

Une autre dimension du problème est l'application de **contraintes** supplémentaires liées à l'état du réseau ou aux préférences de l'utilisateur. Là ou les critères vont orienter la sélection des meilleures solutions parmi les trajets admissibles, les contraintes supplémentaires réduisent la taille de l'ensemble des trajets admissibles de la requête.

Voici une liste de contraintes courantes dans le domaine de la recherche d'itinéraire :

- Interdire un ensemble de **modes** de déplacement (par exemple on recherche les trajets sans train).
- Interdire une **succession particulière de modes** de déplacement (par exemple, il n'est pas pertinent de prendre un véhicule privé tel que la voiture entre deux trains, puisqu'il est peu probable que nous ayons laissé notre véhicule à la bonne gare au préalable).
- Interdire un ensemble de **courses, lignes, arrêts ou exploitants** (pour des raisons d'accidents, de travaux, de visibilité du réseau, d'accessibilité), à certains horaires seulement ou de manière globale.
- Interdire les **montées** ou **descentes** dont la **durée** est inférieure à certaines bornes définies en fonction du mode ou de l'arrêt (parfois même de la course). c'est-à-dire faire en sorte que la durée des changements soit réaliste et non nulle. On appellera ces bornes le **temps de changement minimum** (à l'arrêt, pour le mode), ou la **pénalité de changement**.
- Interdire les **montées** ou **descentes** à certains arrêts pour certains itinéraires seulement.
- Limiter le **nombre total de changements** à une borne supérieure.
- Limiter la **durée d'attente à un arrêt** à une borne supérieure.
- Limiter le **temps total** (ou la distance totale) de marche à pied, vélo, voiture à une borne supérieure ou, plus rarement, inférieure.
- Interdire toute succession de CORRESPONDANCES MARCHE À PIED. En effet, l'une des utilités de définir cet ensemble est d'exclure certaines correspondances des résultats.
- Interdictions de tourner à droite ou à gauche, de faire demi-tour en voiture.

2.4 Variations du problème

Nous présentons dans cette section plusieurs variations sur le thème de la requête d'**arrivée au plus tôt**. D'une part les requêtes multiple **1-n**, **n-1** ou **n-m** permettent de spécifier plusieurs lieux de départ ou d'arrivée, et sont utiles pour combiner des calculs d'itinéraire sur différents graphes. D'autre part, la **profile query** cherche à trouver tous les trajets optimaux de la journée.

Requête 1-n Une **requête 1-n** en *départ à* cherche à obtenir les trajets optimaux pour le critère d'**arrivée au plus tôt** avec l'une des deux contraintes :

$A@\tau \rightarrow (B_1, ..., B_n)$: départ à $A@\tau$ et arrivée à l'un des B_i.

ou

$A@\tau \rightarrow *$: départ à $A@\tau$ et arrivée quelconque.

On cherche ainsi tous les trajets optimaux partant d'un arrêt ou lieu donné à un horaire donné vers tous les autres arrêts.

L'équivalent en *arrivée à* existe aussi.

$* \rightarrow A@\tau$ cherche l'ensemble des trajets arrivant à $A@\tau$ et de départ quelconque (ou dans un ensemble de lieux).

Requête n-m Une **requête n-m** en *départ à* s'écrit

$((A_1@\tau_1), ..., (A_n@\tau_n)) \rightarrow (B_1, ..., B_m)$.

C'est une requête ambigüe car elle peut avoir deux significations. La première est qu'on l'on cherche à obtenir la réponse à toutes les **requêtes 1-m** $A_i@\tau_i \rightarrow (B_1, ..., B_m), \forall i \in [1, n]$, c'est-à-dire tous les trajets optimaux pour tout $A_i@\tau_i \rightarrow B_j$. La deuxième est que l'on cherche à obtenir la meilleure de ces réponses, c'est-à-dire celle dont l'arrivée est au plus tôt. En général, c'est plutôt au deuxième sens que nous nous référons lorsque nous parlons de requête n-m dans ce document.

Profile query Une **requête de profil ou profile query** définie dans [Dell2012a] n'est pas une RI puisque l'on cherche non pas un ou plusieurs trajets mais une fonction de profil.

Étant donné un couple d'arrêts (ou de lieux) (A, B) on cherche à déterminer la fonction $f_{A,B} : \Pi \rightarrow \mathbb{N}$ telle que $\forall \tau \in \Pi, f_{A,B}(\tau)$ est l'horaire d'arrivée optimal de la RI $A@\tau \rightarrow B$.

On peut l'écrire $A@* \rightarrow B$.

3 Modélisation des réseaux

Cette section est consacrée à la description de l'état de l'art concernant les différents modèles de représentation des réseaux de transport rencontrés dans la littérature dans le contexte des problèmes de calcul de plus court chemin. Nous exposerons les modèles utilisés d'un côté pour les réseaux de transport individuel, et de l'autre ceux des réseaux de transport en commun, puis comment combiner ces deux types de modèles.

3.1 Réseaux de transport individuel

Nous nous intéressons ici aux modélisations qui permettent de répondre à la requête d'**arrivée au plus tôt** dans un réseau de voirie, en utilisant les modes de déplacement individuels tels que la marche à pied, le vélo et la voiture.

La modélisation est bien plus simple que celle du réseau de transport en commun car le temps de parcours d'un tronçon de voirie ne dépend pas de l'horaire auquel il est parcouru, contrairement au cas d'une montée en transport en commun, qui dépend de la différence entre l'horaire du prochain passage et l'horaire actuel.

3.1.1 Modèle de graphe

La façon la plus naturelle de modéliser le problème est celle d'un graphe orienté $(\mathcal{N}, \mathcal{A})$ ou $\mathcal{N}$ est l'ensemble des nœuds du graphe et $\mathcal{A}$ l'ensemble des arcs du graphe. Les arcs du graphe représentent les tronçons de voirie, orientés dans le sens de circulation dans le cas routier, et les nœuds du graphe représentent leurs intersections.

Cette modélisation permet d'appliquer des algorithmes de plus court chemin tels que l'algorithme de Dijkstra de manière immédiate, en attribuant à chaque arc un coût égal soit au temps nécessaire pour parcourir le tronçon, soit à sa distance totale.

3.1.2 Séparation en plusieurs graphes

Dans certains travaux, comme [Kirc2013] ou [Geis2010], le réseau de voirie est décomposé en trois réseaux différents : le réseau piéton, le réseau vélo et le réseau voiture. Ils sont modélisés sous la forme de trois graphes orientés séparés, chacun ne considérant que les tronçons compatibles avec le mode spécifié et les intersections entre ces tronçons.

Remarquons qu'il est possible, pour gagner en simplicité et en espace mémoire consommé, d'unifier ces trois graphes en un seul graphe contenant tous les types d'arcs. Chaque arc est alors marqué des modes qu'il est admis d'emprunter. Les contreparties de cette modélisation sont les suivantes :

- augmentation du nombre de nœuds et d'arcs par rapport à chaque graphe séparé,
- condition supplémentaire lors de l'évaluation d'un arc,
- certaines méthodes d'accélération d'algorithme de recherche de plus court chemin ne peuvent pas s'appliquer.

3.1.3 Prise en compte des arrêts et des points d'intérêt

Les arrêts et les points d'intérêts sont des lieux géographiques particuliers, qui peuvent servir de point de départ ou d'arrivée d'un trajet multimodal. En général, ces lieux ne sont pas directement connectés au réseau de voirie, mais on connait leur position géographique. Une question importante est donc de les intégrer au graphe des voiries.

Une manière classique de le faire consiste à chercher le tronçon géographique le plus proche t, ce qui est un problème d'optimisation géographique. Puis il faut connecter le nœud représentant le lieu géographique avec les arcs représentant le tronçon trouvé. La littérature propose plusieurs méthodes de connexion : relier le nœud au nœud le plus proche ([Pajo2009]), ou dupliquer et découper les arcs du tronçon pour insérer le nœud entre les arcs ainsi créés ([Bous2010]) en sont deux exemples. La figure 2 illustre le type de graphe obtenu avec cette dernière approche.

Plus de détails sur cette problématique sont disponibles en annexe A du document, consacrée à l'accrochage des lieux et leur modélisation dans les graphes individuels.

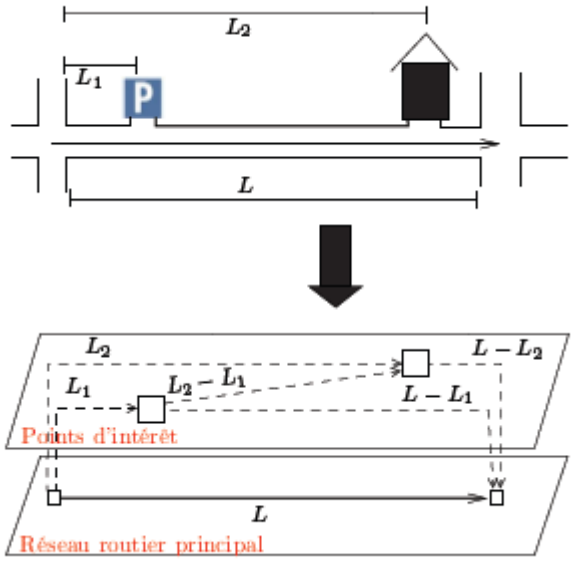


FIGURE 2 – Deux points d'intérêt sur le même tronçon de voirie ([Bous2010])

3.1.4 Prise en compte des pistes cyclables

L'intégration des pistes cyclables dans le réseau de voirie est plus ou moins complexe, en fonction de la qualité des données et de leur prétraitement. Le problème est similaire à celui des points d'intérêt, à ceci près qu'il faut non pas connecter des lieux géographiques, mais des tronçons cyclables bien souvent distincts du réseau de voirie classique.

Dans [Sauv2011], les données sont issues d'OpenStreetMap mais préalablement intégrée par un collecteur de terrain, qui vérifie et rajoute les pistes cyclables sur OSM

et les relie à une base de données privée. L'algorithme découpe ensuite les données à chaque intersection de rue ou de pistes cyclables et crée autant d'arcs que nécessaire. Un problème qui peut se poser est que bien souvent les pistes cyclables sont tracées en parallèle des routes et n'y sont pas connectées, c'est pour cette raison qu'il faut vérifier manuellement les données d'entrée ou de sortie.

3.1.5 Prise en compte du trafic temps réel

Le temps réel transforme le graphe indépendant du temps en graphe dépendant du temps. Le temps consommé pour le parcours d'un arc dépend alors du temps au nœud de départ de cet arc. Dans [Geis2010], ceci est modélisé comme une fonction dépendant de l'arc $f : \Pi \to \mathbb{R}^+$ qui à chaque $\tau \in \Pi$ associe le temps nécessaire pour parcourir cet arc.

Il est important de noter qu'il faut que la contrainte FIFO soit respectée pour la fonction f, c'est-à-dire que partir plus tard ne permette pas d'arriver plus tôt : $\forall \tau' > \tau \in \Pi, \tau' + f(\tau') \geq \tau + f(\tau)$. Dans le cas contraire, les algorithmes de plus court chemin tels que l'algorithme de Dijkstra ne sont plus exacts.

3.1.6 Prise en compte des manœuvres interdites

Les interdictions de tourner sont traitées dans la littérature de deux façons différentes. Soit par l'extension du graphe, en particulier en ajoutant des nœuds aux intersections, soit par la mise en place et l'exploitation du graphe adjoint (ou line graphe, ou graphe dual) : graphe qui inverse les arcs et les nœuds.

Des travaux complets résolvant cette question et celle des délais aux carrefours pour des successions de plus de deux arcs peuvent être trouvés dans la thèse de [Bous2010]. Elle introduit une méthode d'extension du graphe en le composant avec un automate représentant les restrictions de manœuvres.

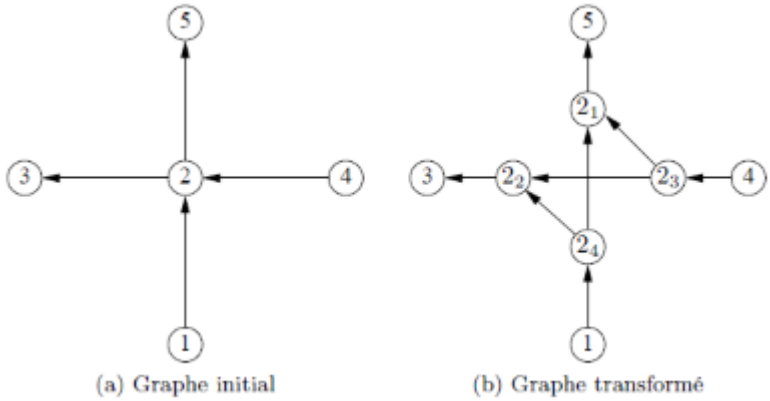
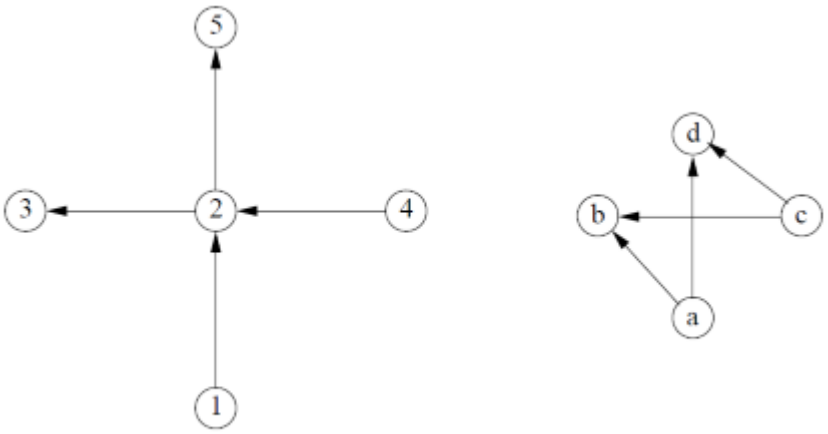


FIGURE 3 – Manœuvres interdites : graphe étendu ([Sauv2011])



(a) Graphe initial (b) Graphe adjoint

FIGURE 4 – Manœuvres interdites : graphe adjoint ([Sauv2011])

3.2 Transport en commun

Nous nous intéressons ici aux modélisations qui permettent de répondre à la requête d'**arrivée au plus tôt** dans un réseau de transport en commun.

La difficulté principale par rapport à la modélisation du réseau de voirie est que tout chemin n'est pas seulement spatial mais aussi temporel, et que cette dimension doit être prise en compte dans l'algorithme de résolution.

L'idée principale reste de modéliser le réseau sous forme de graphe pour pouvoir y appliquer des algorithmes de calcul de plus court chemin.

3.2.1 Graphe *time-dependant*

L'idée est de modéliser le réseau dépendant du temps sous forme de graphe, de manière similaire au graphe routier : chaque nœud représente un lieu géographique. Le graphe est accompagné d'une fonction d'évaluation des coûts dont les paramètres sont l'arc à évaluer et un entier représentant le temps. Cette façon de modéliser les réseaux dépendant du temps a été analysée par Orda et Rom dans [Orda1991] et utilisée pour représenter des tables horaires dans [Brod2003].

Nous décrivons ici la modélisation de Geisberger dans [Geis2010].

Pour chaque itinéraire $I = (n, (A_i)_{i\in[1,n]}) \in \mathcal{I}$, et pour chaque arrêt A_i de l'itinéraire I, un **nœud sur itinéraire** $A_{i,I}$ est créé. Tous les nœuds $(A_{i,I})_{i\in[1,n]}$ d'un itinéraire sont reliés entre eux par des arcs orientés dans l'ordre de l'itinéraire. A chaque arc d'itinéraire est associé une fonction d'évaluation du temps de parcours directement lié aux étapes élémentaires représentées par cet arc.

Pour chaque arrêt $A \in \mathcal{A}$, un **nœud de transfert** A_t est créé. Chaque nœud de transfert A_t est relié à tous les nœuds sur itinéraire $A_{i,I}$ du même arrêt par deux arcs, l'un représentant la montée et l'autre la descente. Les temps de montée et descente

peuvent être nuls, dépendre de l'arrêt ou dépendre de l'itinéraire.

Enfin, chaque correspondance marche à pied entre deux arrêts est modélisée par deux arcs entre les **nœuds de transfert** des arrêts en correspondance. Le coût de ces arcs est lié au temps de parcours estimé entre chaque arrêt.

On peut visualiser ce modèle dans la figure 5 reprenant l'exemple du réseau de Colmar présenté en section 1.

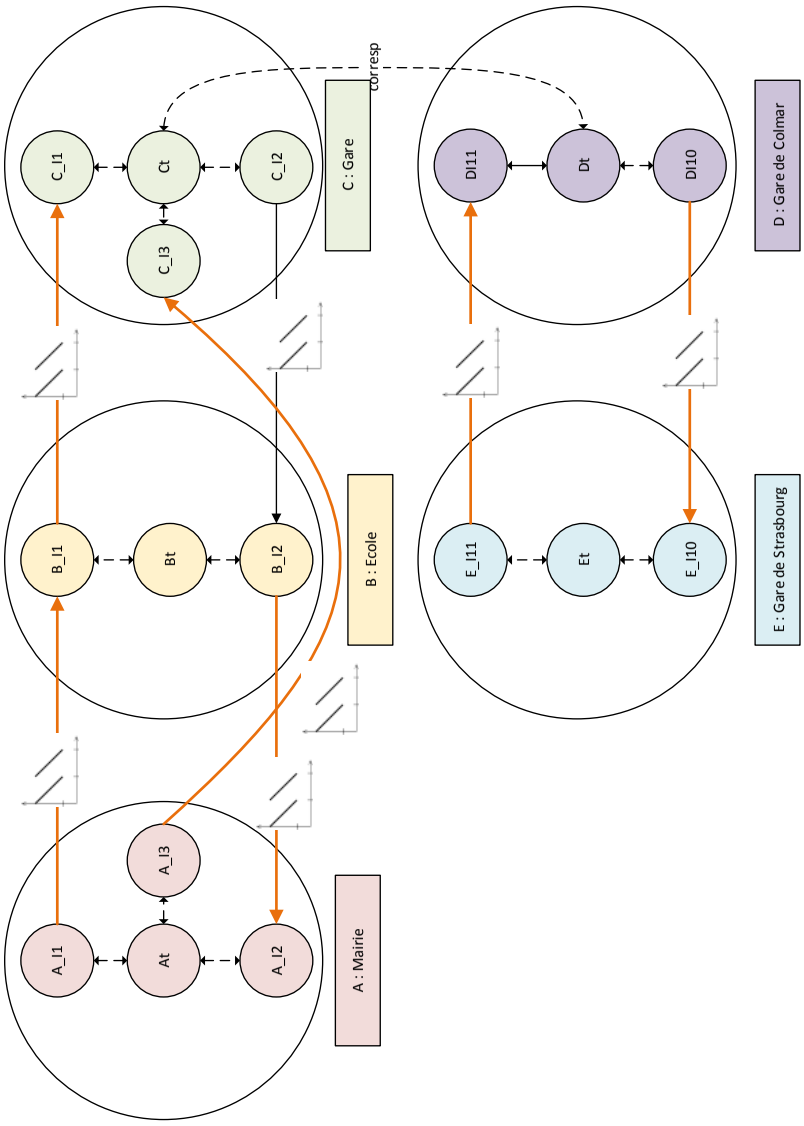


FIGURE 5 – Modèle *time-dependant*

3.2.2 Variantes du graphe *time-dependant*

Pour un modèle plus proche de la réalité, on trouve dans [Schn2009] deux variantes du modèle dépendant du temps. Notons d'abord que les arcs de descente de transport possèdent un coût afin de forcer un temps de changement minimum à une station mais que les coûts des arcs de montée sont toujours nuls (pour optimiser le temps de calcul multicritère).

La première variante concerne la modélisation des **correspondances marche à pied**. Plutôt que de partir du nœud de station, on rajoute un nœud *foot* F, relié par des arcs à coûts nuls à tous les autres nœuds de l'arrêt (desquels il est autorisé de descendre). Les temps de changement nécessaire pour descendre d'un train à une gare et monter dans un autre train à une autre gare sont alors intégrés dans les coûts de marche à pied entre les nœuds F de chaque station . Ceci est illustré dans la partie gauche de la figure 6.

Des **temps de changement spécifiques** sont également rajoutés au modèle. Deux trains partageant le même quai peuvent avoir un temps de changement réduit par exemple. Pour chaque règle de changement spécifique, il faut modifier le graphe. Plus précisément, s'il est possible d'aller du train t1 au train t2 en un temps de changement particulier, il faut rajouter un arc direct entre les nœuds d'itinéraire de t1 et de t2 qu'il n'est possible d'explorer qu'après être arrivé par t1, et interdire la descente ou la montée du nœud de t2 si t1 est la dernière course empruntée. Cet arc spécial est illustré sur la partie droite de la figure 6.

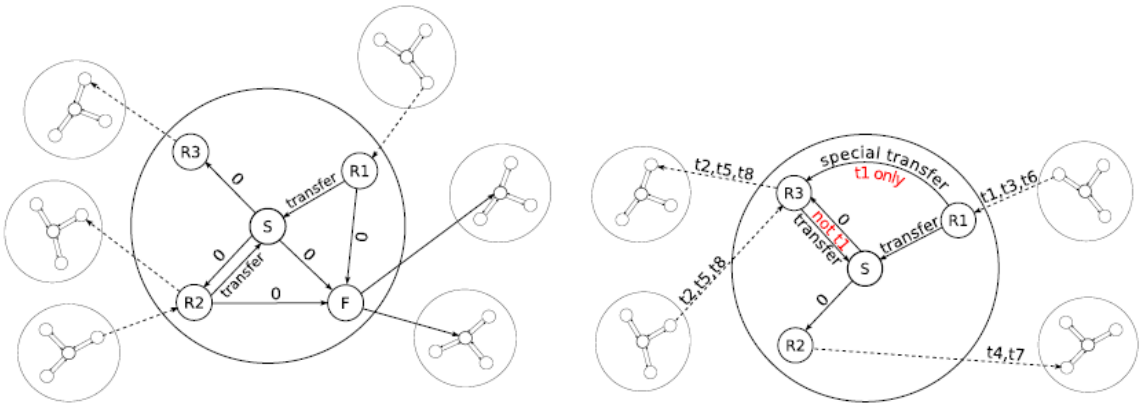


FIGURE 6 – nœud MARCHE et changements spéciaux ([Schn2009])

3.2.3 Graphe *time-expanded*

La première variante de cette modélisation, que nous ne détaillons pas, a été introduite par Pallotino et Scutella dans [Pall1998]. Elle consiste étant donné un ensemble discret d'horaires de la journée à exploser chaque arrêt du réseau en autant de nœuds qu'il y a d'horaires dans Π, puis à relier les nœuds entre eux pour modéliser les courses d'un arrêt à un autre ainsi que l'attente à un arrêt.

Nous décrivons ici la modélisation de Geisberger ([Geis2010]), très proche de celle de Scutella ([Pall1998]). L'auteur utilise non pas un horaire de passage, mais des horaires de départ et d'arrivée à chaque arrêt, ce qui est par ailleurs transparent avec la représentation du tableau horaire par les **étapes élémentaires** introduites précédemment.

Un graphe *time-expanded* a trois types de nœuds : nœud de transfert, nœud de départ et nœud d'arrivée. Chaque nœud porte une information temporelle et spatiale (horaire et arrêt), et représente donc un évènement.

Pour chaque étape élémentaire $(\mathsf{C}, \mathsf{A}_1@\mathsf{h}_1, \mathsf{A}_2@\mathsf{h}_2)$, un nœud de départ $\mathsf{A}_1\mathtt{d}@\mathsf{h}_1$ modélise l'arrêt A_1 au temps de départ h_1, un nœud d'arrivée $\mathsf{A}_2\mathtt{a}@\mathsf{h}_2$ modélise l'arrêt A_2 au temps d'arrivée h_2 et un arc $\mathsf{A}_1\mathtt{d}@\mathsf{h}_1 \to \mathsf{A}_2\mathtt{a}@\mathsf{h}_2$ modélise le passage d'un évènement à l'autre par l'étape élémentaire. Si la course repart de A_2 au temps h_3, un nœud et un arc $\mathsf{A}_2\mathtt{a}@\mathsf{h}_2 \to \mathsf{A}_2\mathtt{d}@\mathsf{h}_3$ modélisent l'attente du véhicule à l'arrêt A_2. Ce nœud et cet arc existent même si $\mathsf{h}_2 = \mathsf{h}_3$.

Pour chaque nœud de départ $\mathsf{A}\mathtt{d}@\mathsf{h}$, un nœud de transfert $\mathsf{A}\mathtt{t}@\mathsf{h}$ au même horaire est ajouté et un arc $\mathsf{A}\mathtt{d}@\mathsf{h} \to \mathsf{A}\mathtt{t}@\mathsf{h}$ modélise la montée dans le transport. S'il existe d'autres nœuds de transfert à des temps ultérieurs $\mathsf{A}\mathtt{t}@\mathsf{h}'$, un arc $\mathsf{A}\mathtt{t}@\mathsf{h} \to \mathsf{A}\mathtt{t}@\mathsf{h}'$ relie le nœud de transfert de h' minimum. Ces arcs forment la *chaîne d'attente* à l'arrêt A.

Enfin, pour permettre un changement après avoir atteint un nœud d'arrivée $\mathsf{A}\mathtt{a}@\mathsf{h}$, un arc du nœud vers le premier nœud de transfert $\mathsf{A}\mathtt{t}@\mathsf{h}'$ tel que $\mathsf{h}' \geq \mathsf{h} + \mathit{transfert}(\mathsf{S})$ est créé. Un arc est créé de la même façon pour tous les arrêts en correspondance avec A qui possède un nœud de transfert $\mathsf{B}\mathtt{t}@\mathsf{h}'$ avec $\mathsf{h}' - \mathsf{h}$ suffisamment grand pour permettre la marche.

Ce modèle, appliqué à l'exemple du réseau de Colmar présenté en section 1, est illustré dans la figure 7.

3.2.4 Graphe par arrêt

Pour un graphe direct acyclique, tout algorithme glouton de calcul de plus court chemin fonctionne parce que tout sous-chemin d'un plus court chemin est un plus court chemin. Dans un graphe multimodal dépendant du temps ou chaque station est représentée par un nœud, cela n'est plus vrai. En effet, pour des raisons d'horaires, il peut être optimal d'aller de A vers C en CAR, en passant par B, mais pas d'aller de A vers B en CAR plutôt qu'en BUS.

C'est pour cette raison que le graphe *time-dependant* duplique les nœuds par itinéraire, ce qui, combiné au fait qu'aucune course n'en dépasse une autre sur un même itinéraire (FIFO), valide la propriété *tout sous-chemin d'un plus court chemin est un plus court chemin.*

Berger introduit dans [Berg2009] un modèle compact plus simple que les deux modèles précédents. C'est également un graphe $\mathsf{G} = (\mathcal{A}, \mathsf{E})$ mais dont les nœuds représentent des stations et ne sont pas dupliqués ni par itinéraire ni par course. Elle introduit également les « étapes concaténables » qui représentent deux étapes élémentaires effectuant un changement à une station et dont le temps de changement est supérieur à une valeur minimale de changement. Les arcs sont parallèles entre les stations : un arc est

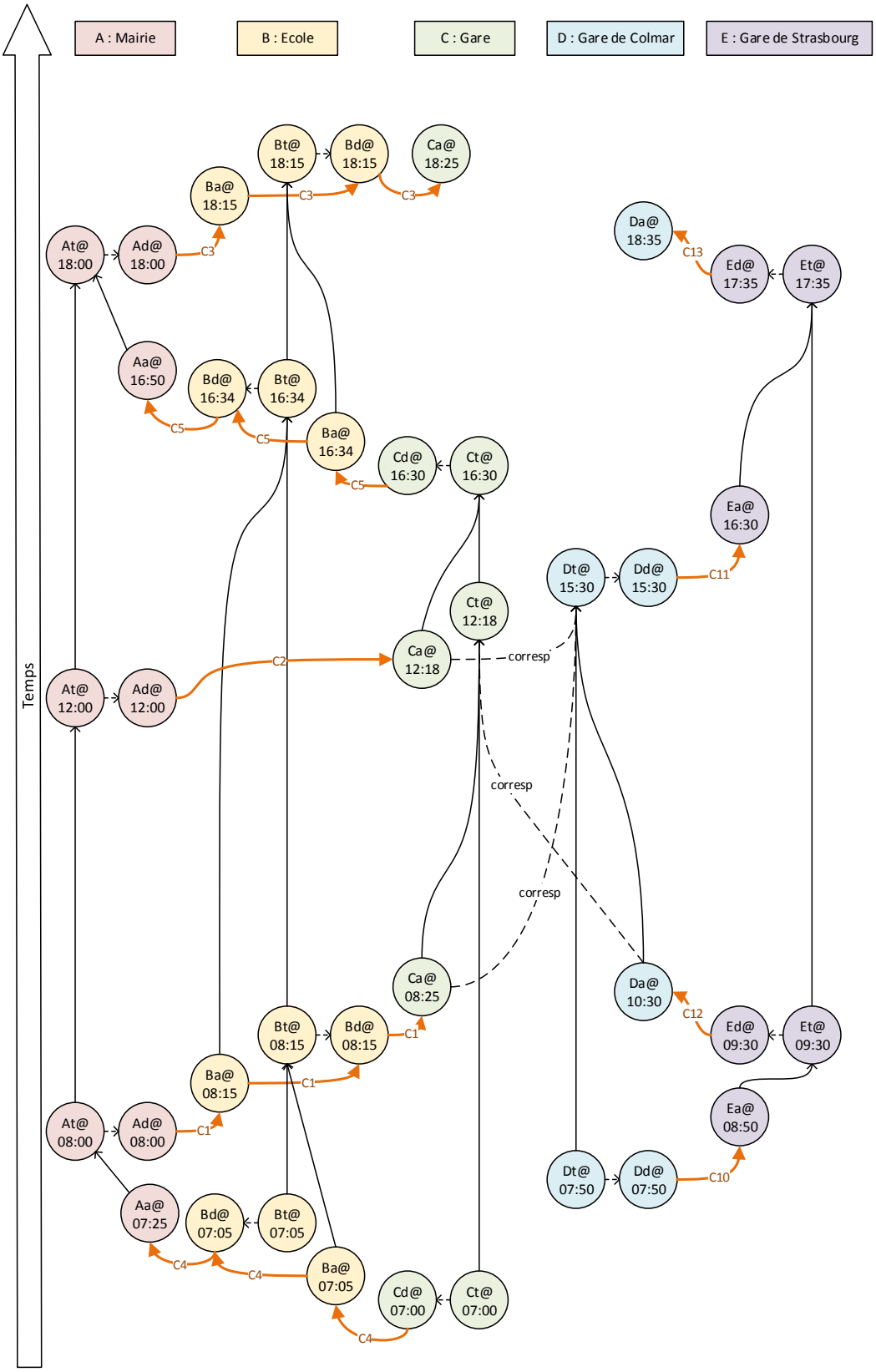


FIGURE 7 – Modèle *time-expanded*

inséré par itinéraire. Le but est d'appliquer les méthodes d'optimisation de contraction et arc-flags utilisées en recherche d'itinéraire sur graphe de voirie.

Geisberger réutilise ce modèle en le simplifiant encore dans [Geis2010] afin de limiter le nombre d'arcs lors de la contraction du graphe. Il n'y a plus qu'un arc reliant deux stations, ce qui élimine la condition FIFO du graphe. L'algorithme est modifié en conséquence pour prendre cela en compte.

3.2.5 Modélisation des grandes stations

Plusieurs approches sont possibles. Dans [Stra2012], deux approches sont énoncées :

- Un seul arrêt pour une grande station avec un temps de changement minimum (dans la station) égal au temps le plus élevé nécessaire pour passer d'un quai à un autre.
- Modéliser un arrêt par quai, avec des correspondances marche à pied entre chaque quai reflétant leur éloignement.

Le problème de la première approche est que le pire temps de changement d'une gare est souvent assez élevé, alors que passer d'un quai au quai d'en face est quasiment immédiat. Le problème de la deuxième approche est que le nombre d'arcs du graphe augmente considérablement et que des informations précises de temps de marche entre chaque quai ne sont pas souvent connues. Enfin certains transporteurs changent régulièrement les trains de quais.

3.2.6 Prise en compte de courses cycliques

Les courses peuvent contenir deux fois le même arrêt à deux horaires distincts. Geisberger ([Geis2010]) introduit une notion supplémentaire d'*évènement à l'arrêt* pour les départs et les arrivées dans sa définition des connections élémentaires. Dans la modélisation *time-expanded*, les nœuds sont distincts pour des horaires distincts de toute façon. Dans la modélisation *time-dependant*, les nœuds représentant le même arrêt à deux moments différents dans l'itinéraire d'un train sont dupliqués autour du nœud de transfert.

3.2.7 Prise en compte d'arrêts prolongés à l'arrêt

La modélisation par étape élémentaire permet à chaque course d'avoir deux horaires par arrêt, un d'arrivée à l'arrêt, l'autre de départ. D'autres modèles simplifiés ne le font pas nécessairement. En effet c'est une composante importante dans le cas des trains, mais quasiment inexistante pour les autres modes de transport en commun.

3.2.8 Prise en compte des trains de nuit et de la périodicité du réseau

Un problème récurrent en informatique est que la mémoire n'est pas illimitée. De ce fait, un réseau périodique illimité dans le temps ne peut pas être représenté comme

tel en mémoire. Les modélisations précédemment abordées considèrent que le jour de fonctionnement est fixé et que les horaires s'étalent en général sur 24h. Dans ce type de modélisation, il n'est pas possible de prendre en compte certains trains de nuit et la continuation d'un trajet au lendemain, où les horaires et les courses auront peut-être changé, est impossible.

Pour résoudre ce problème, certains travaux (comme [Schn2009]) considèrent les réseaux périodiques de période Π (1440 minutes) ou chaque course est répétée à l'identique d'un jour à l'autre. Il est alors possible de rajouter des arcs dans le graphe *time-expanded* représentant l'attente entre la dernière course de la journée et la première de la journée suivante.

Dans le cas plus réaliste où les courses ne sont pas identiques mais suivent un schéma déterminé par les jours de la semaine, il suffit de conserver une donnée indiquant le jour de la semaine dans les labels calculés et de marquer chaque arc par des autorisations et interdictions de les parcourir pour chaque jour de la semaine.

Dans le cas d'une modélisation *time-dependant*, l'accès aux horaires de la journée suivante peut se faire à condition d'avoir une structure de données horaire à la fois suffisamment compacte pour donner accès à plusieurs jours différents dans le même calcul et dont le temps d'accès est en O(1).

3.2.9 Prise en compte des risques de retard

Dans [Schn2009], il est expliqué comment mettre à jour le graphe *time-expanded* en fonction des horaires en temps réel de chaque course. Les arcs de changement d'arrêt doivent maintenir les temps de changement minimaux et donc potentiellement changer de nœud de fin, ce qui remodèle complètement le graphe. Le changement est plus important que dans le cas *time-dependant*, où seules les fonctions d'évaluation de coût de chaque arc changent, mais la structure du graphe par itinéraire reste inchangée.

Dans [Stra2012], Strasser définit un modèle stochastique simulant des retards éventuels pour pallier à un manque de données réelles. Nous n'entrerons pas dans les détails de cette modélisation, très éloignée du cadre de nos travaux.

3.3 Combinaison des réseaux

Comme on l'a vu, chaque réseau, transport en commun et routier, comporte des spécificités qui se modélisent de manière différente dans un cas ou dans l'autre. Or, un trajet multimodal doit pouvoir contenir à la fois de la marche à pied, du tram et du vélo par exemple. Il est donc nécessaire de combiner les réseaux en un seul réseau multimodal. Deux méthodes principales sont utilisées dans la littérature. D'une part, il est possible de laisser chaque graphe intact et d'utiliser des algorithmes de plus court chemin distribué sur plusieurs graphes. D'autre part, nombre de travaux sur la recherche d'itinéraire multimodale font le choix de fusionner les graphes en un seul graphe.

Dans ces deux cas, il est nécessaire de résoudre une forme du problème de recherche des plus proches voisins pour trouver les tronçons de voirie à proximité des arrêts de

transport en commun et de modéliser plus ou moins finement les liens possibles entre les réseaux.

Précisons que nombre de travaux n'intègrent pas de graphe des voiries dans le calcul, et considèrent un ensemble prédéfini de correspondances marche à pied possibles entre arrêts. C'est le cas de la plupart des travaux traitant de réseaux ferroviaires, comme [Schu2005], [Geis2010] ou [Schn2009].

3.3.1 Réseaux combinés en un seul graphe

Dans la plupart des travaux traitant de recherche d'itinéraire multimodal (par exemple : [Pajo2009], [Pajo2013], [Kirc2013] ou [Grab2010]), l'ensemble des graphes représentant les réseaux de transport en commun et de voirie sont combinés pour ne former qu'un seul graphe. Deux exemples sont visibles dans la figure 8.

Les arcs du graphe sont alors typés par le mode utilisé et les transitions d'un mode à l'autre sont contraintes par des règles métier. Il est possible de modéliser ces règles en considérant la succession de modes comme un mot qui doit appartenir à un langage. Par exemple, si l'on exclut les taxis, un trajet combinant la voiture, puis le bus, puis la voiture à nouveau est interdit, puisqu'il est impossible de reprendre une voiture en sortant d'un bus. Le mot (VP,TC,VP) ne fait pas partie du langage autorisé. Cette contrainte de langage est modélisée avec un automate, approche introduite par Lozano et Storchi dans [Loza2001]. Nous ne rentrerons pas dans les détails de résolution d'un problème de recherche de plus court chemin à contraintes par langage régulier, mais d'importants travaux et de nombreuses références sur le sujet se trouvent dans les travaux de Artigues et Huguet dans [Arti2013], ainsi que dans la thèse de Kirchler [Kirc2013].

Dans [Liu2011], ce n'est pas un automate mais une « *Switch Point Matrix* » qui détermine les transitions de modes possibles entre les graphes : cette matrice indique quels nœuds permettent de passer d'un graphe à l'autre, et sous quelles conditions. De plus la modélisation du graphe de transport en commun est légèrement différente, puisqu'elle est considérée comme indépendante de l'horaire avec des coûts fixes par arc, mais prenant en compte des coûts d'attente dépendant du temps aux divers *switch points* permettant de transiter d'un mode transport à l'autre.

3.3.2 Réseaux séparés en plusieurs graphes et algorithmes distribués

L'alternative à la création d'un graphe unique à plusieurs couches est d'appliquer des algorithmes de recherche de plus court chemin dans un graphe distribué en classes. L'un des premiers algorithmes de résolution en temps polynomial est décrit dans [Wang2004].

Chaque nœud du graphe appartient à une ou plusieurs classes, comme le montre la figure 9 et chaque réseau correspond à une seule classe. Chaque arc du graphe appartient à un unique réseau. Chaque réseau connait ses propres nœuds, ses propres arcs, et les coûts liés à ses arcs ; de plus il est muni d'un système d'information capable de rechercher un plus court chemin en son sein. Le problème est alors de définir un algorithme central, connaissant les points d'intersections des réseaux, capable de composer des requêtes

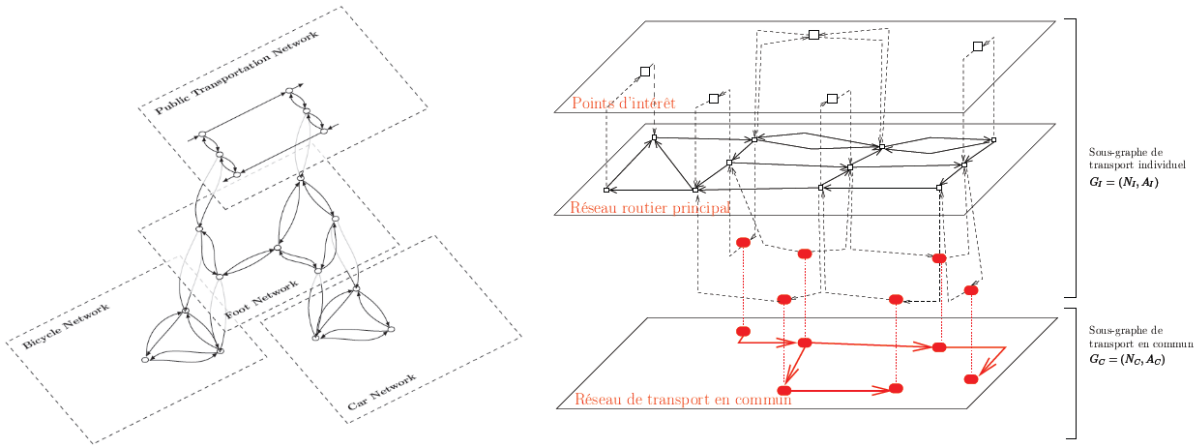


FIGURE 8 – Modèles de graphe combiné de Kirchler (à gauche) et Bousquet (à droite)

à chaque sous-système et de trouver un plus court chemin global entre deux points quelconques du graphe distribué.

Ce type d'algorithmes est utilisé par les systèmes de calculs d'itinéraire distribué sur plusieurs sources de données de transport en commun, comme DELFI/EU-Spirit en Allemagne ou Transport Direct aux Royaumes-Unis.

Ces algorithmes sont améliorés pour le calcul parallèle de plus court chemin horaire dans un réseau multimodal dans les thèses de Kamoun [Kamo2007], Feki [Feki2010], et Ayed [Ayed2011].

Enfin, le calculateur de Cityway utilise une variante simplifiée de cette méthode pour permettre de trouver les plus courts chemin avec départ ou arrivée en marche à pied, vélo ou véhicule privé.

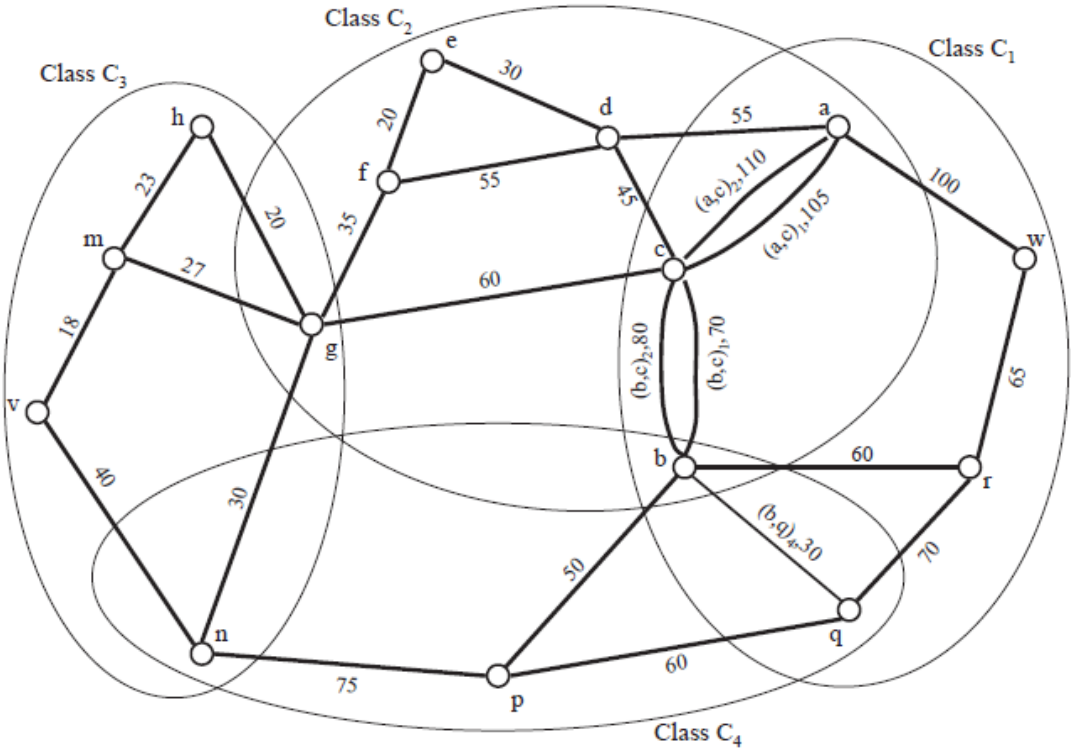


Fig. 1. $G(V,E)$ for a distributed system with four classes.

FIGURE 9 – Extrait de [Wang2004]

Chapitre II

État de l'art sur les algorithmes de résolution

Sommaire

Ce chapitre est consacré à l'exploration de l'état de l'art sur la recherche de plus court chemin dans un graphe. Les algorithmes fondateurs de Bellman-Ford et Dijkstra sont décrits dans la section 2. Diverses optimisations applicables quand les poids des arcs sont indépendants du temps sont présentées dans section 3. Enfin, nous présentons les algorithmes plus spécifiques aux réseaux de transport en commun dans la section 4.

1 Introduction

Étant donné un graphe $\mathcal{G}(\mathsf{N}, \mathsf{A})$ orienté ou non, avec N l'ensemble de ses nœuds et A celui de ses arcs pondérés. Les algorithmes, issus de la littérature, présentés dans cette section permettent de résoudre le problème du plus court chemin dans $\mathcal{G}$, c'est-à-dire, pour deux nœuds P et Q de déterminer le chemin de poids minimum allant de P vers Q.

Lorsque les poids des arcs représentent un temps, résoudre un problème de plus court chemin revient effectivement à résoudre le problème du trajet le plus rapide. Ce dernier équivaut au problème de l'**arrivée au plus tôt** si l'on a fixé un horaire de départ, et à celui du **départ au plus tard** lorsque l'on a fixé un horaire d'arrivée.

Pour plus de détails et des résultats d'expérimentations comparant temps de prétraitement, utilisation d'espace mémoire et temps de calcul, nous orientons le lecteur vers l'article « *Route Planning in Transportation Networks* » [BDGM2016] de Bast *et al.*

Celui-ci présente une synthèse très complète des principaux algorithmes de résolution des problèmes de calcul d'itinéraire sur des réseaux routiers et de transport en commun.

2 Algorithmes fondateurs

Nous présentons ici les deux algorithmes fondateurs de la recherche de plus court chemin dans un graphe, celui de Bellman-Ford et celui de Dijkstra. Une bonne compréhension de ceux-ci est nécessaire car la majorité des algorithmes de recherche de plus court chemin trouvés dans la littérature héritent de ou utilisent d'une manière ou d'une autre l'un de ces deux algorithmes. En particulier, le calculateur de Cityway utilise une implémentation de l'algorithme de Dijkstra.

2.1 Bellman-Ford

Dans [Bell1956], Bellman et Ford introduisent un des premiers algorithmes de résolution des problèmes de plus court chemin. Il s'appuie sur la programmation dynamique ([Bell1957]), c'est-à-dire sur le principe que l'on peut résoudre un problème d'optimisation en le décomposant en sous-problèmes locaux, et en résolvant à l'optimum ces derniers.

Le sous-problème est le suivant : si l'on connait tous les plus courts chemins de taille inférieure à un entier k, partant de la source P et allant vers un nœud R, comment calculer tous les plus courts chemins de taille inférieure à $k+1$? La réponse est la suivante : pour chaque nœud R, notons $P \rightarrow_k R$ ces chemins. Soit R quelconque, $P \rightarrow_{k+1} R$ est le chemin de coût minimal parmi d'une part $P \rightarrow_k R$, et d'autre part les chemins composés s'écrivant $(P \rightarrow_k S, S \rightarrow R)$ tels qu'il existe un arc (S, R) dans A.

De cela découle naturellement l'algorithme de Bellman-Ford, dont le pseudo-code reporté dans l'algorithme 1 permet de calculer la distance minimale entre P et tous les nœuds du graphe R, que l'on note $d_P(R)$. Une fois que l'on a trouvé tous les chemins optimaux partant de P, on a également trouvé le chemin optimal de P vers Q.

Cet algorithme a également l'avantage de détecter la présence de circuits absorbants, qui rendent le problème insoluble. En effet, si à la fin du calcul, il existe un arc pour lequel une nouvelle application de la boucle ferait diminuer la distance, c'est qu'il existe un circuit absorbant.

La complexité de cet algorithme est de $\mathcal{O}(|N| \cdot |A|)$.

2.2 Dijsktra

L'algorithme de Dijkstra porte le nom de son créateur, qui le publie en 1959 dans [Dijk1959]. Il reste à ce jour l'un des algorithmes de plus court chemin les plus utilisés, en raison de sa simplicité et de sa performance, meilleur que l'algorithme de Bellman-Ford en complexité algorithmique. Il permet de construire l'arbre des plus courts chemins en partant d'une source P dans le graphe et allant vers tous les autres nœuds accessibles

Algorithm 1 Algorithme de Bellman-Ford

```
pour tout R ∈ N, d_P(R) ← +∞
d_P(P) ← 0
for k = 1 jusqu'à k = |N| do
  for all a = (U, V) ∈ A do
    d_P(V) ← min(d_P(V), d_P(U) + poids(a))
  end for
end for
for all a = (U, V) ∈ A do
  if d_P(U) + poids(a) < d_P(V) then
    return Circuit absorbant détecté
  end if
end for
return d_P
```

depuis la source. En particulier figure dans cette construction le plus court chemin de P vers Q.

Dijkstra remarque d'abord que si R est un nœud sur le chemin le plus court de P à Q, la connaissance de ce dernier donne également la connaissance du plus court chemin de P à R. L'algorithme construit donc tous les chemins minimaux qui partent de P, et s'arrête lorsque le chemin minimal vers Q est construit.

L'ensemble des nœuds et l'ensemble des arcs sont partitionnés comme suit :

- N_A les nœuds vers lesquels le plus court chemin depuis P est connu, initialisé à $\emptyset$,
- N_B les nœuds connectés par un arc à l'un des nœuds de N_A, mais qui n'y sont pas eux-mêmes, initialisé à $\emptyset$,
- N_C les autres nœuds, initialisé par tous les nœuds du graphe.
- I les arcs présents dans un plus court chemin de P vers un élément de N_A, initialisé à $\emptyset$,
- II les arcs menant d'un nœud de N_A à un nœud de N_B, un et un seul arc par nœud de N_B, initialisé à $\emptyset$,
- III les arcs rejetés ou non examinés, initialisé avec tous les arcs du graphe.

On transfère P de N_C vers N_A, et l'algorithme se fait ensuite en deux étapes qui se répètent jusqu'à ce que Q soit dans N_A.

La première consiste à examiner chaque arc r du nœud que l'on vient de transférer dans N_A vers un nœud R. Si R était déjà dans N_B, on vérifie si r permet un chemin plus court et on met à jour II le cas échéant. Si R est dans N_C, on met r dans II et R dans N_B.

La deuxième étape consiste considérer les distances des chemins construits de P vers les éléments de N_B. A chaque élément de N_B correspond un chemin unique et une distance : on prend un chemin avec la plus petite distance, on transfère le nœud final de ce chemin de N_B vers N_A et l'arc correspondant de II vers I.

Le calculateur de Cityway utilise cet algorithme, appliqué aux réseaux de transport en commun et de voiries pour trouver les itinéraires optimaux, comme décrit dans le chapitre IV, dans la section 2 « Algorithme monolabel ».

La complexité de cet algorithme dépend de l'efficacité de la fonction qui trouve le chemin de plus petite distance vers les nœuds de N_B à la deuxième étape. Cet algorithme, implémenté en utilisant un tas binaire, a une complexité en $\mathcal{O}(|A| + |N| \cdot \log(|N|))$.

3 Optimisations du calcul du plus court chemin

Nous supposons que les poids des arcs du graphe ne dépendent pas du temps. L'application de l'algorithme de Dijkstra sur un tel graphe permet de trouver le plus court chemin suffisamment rapidement pour des petites instances. Toutefois, pour les plus grands instances, de nombreuses améliorations ont été apportées pour accélérer l'algorithme, avec ou sans prétraitement. Nous détaillons ici les plus courantes.

3.1 Dijkstra bidirectionnel

Dans son livre [Pohl1969], Pohl introduit une première amélioration de l'algorithme de Dijkstra. Rappelons que celui-ci parcourt « aveuglément » le graphe en calculant tous les chemins minimaux vers chaque nœud de celui-ci, trouvés dans l'ordre de distance totale croissante. Dans un graphe modélisant un réseau routier, il est souvent possible de partir dans toutes les directions à partir d'une intersection. L'algorithme de Dijkstra parcourt donc une sorte de disque autour du point d'origine.

Or, le nœud de destination étant connu, il est possible d'utiliser l'algorithme de Dijkstra en partant de celui-ci et en parcourant les arcs dans le sens inverse. L'idée générale de la recherche bidirectionnelle est d'alterner ces deux applications de l'algorithme. Notons A_d, B_d les ensembles N_A, N_B (voir description de l'algorithme de Dijkstra) du calcul direct, et A_r, B_r ces ensembles pour le calcul inverse. Lorsque un nœud R est dans $A_d \cap A_r$, on sait que $P \to R$ et $R \to Q$ sont des chemins minimaux, mais pas nécessairement que leur combinaison est minimale. Il faut retenir et mettre à jour μ, le minimum, pour tout $R \in A_d \cap A_r$, de la somme des distances de ces deux chemins.

La condition d'arrêt de l'algorithme est alors que les deux nœuds de distance minimum des deux ensembles B_d et B_r soient à une distance respectivement de P et de Q supérieure à μ. Alors, il ne peut exister de chemin plus court que μ reliant P et Q.

L'avantage de cette méthode est que l'on a exploré deux disques de rayon $\mu/2$ plutôt qu'un disque de rayon μ autour de P, réduisant l'espace de recherche, comme le montre la figure 10, extraite d'une présentation par Goldberg, Harrelson, Kaplan et Werneck dans le cadre de leurs travaux sur l'algorithme ALT ([Gold2005]).

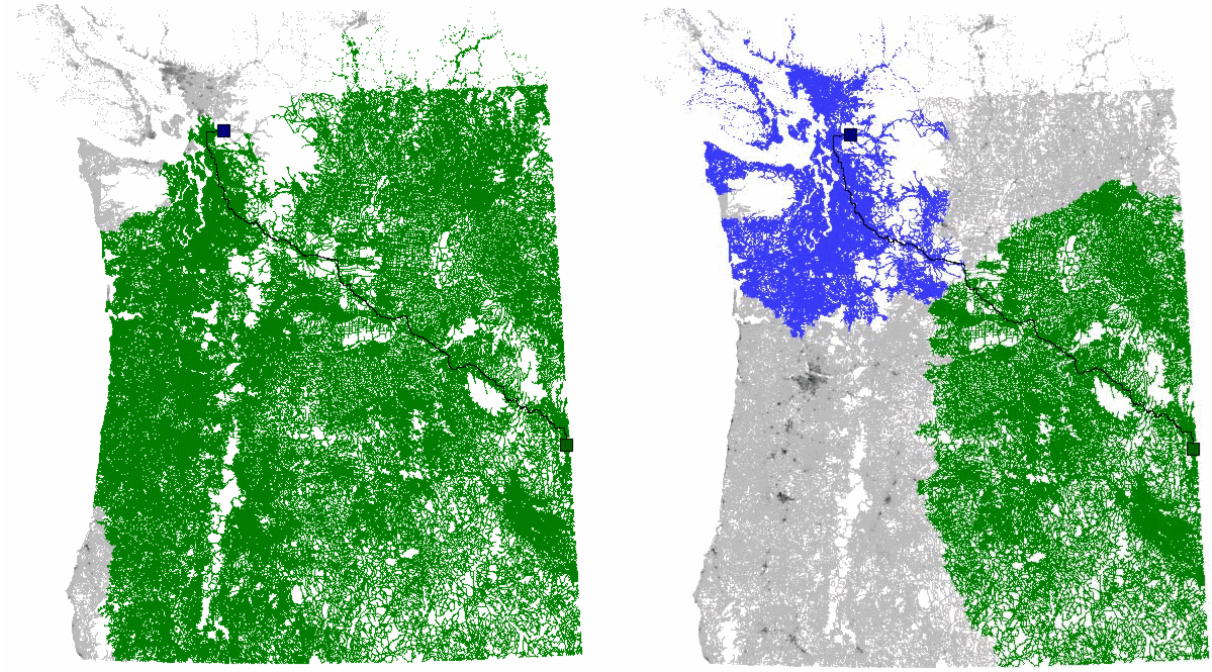


FIGURE 10 – À gauche Dijkstra unidirectionnel, à droite Dijkstra bidirectionnel (Goldberg et al.)

3.2 Algorithme A*

Une autre approche d'optimisation de Dijkstra, que l'on peut par ailleurs combiner avec l'approche bidirectionnelle, est d'utiliser une fonction d'estimation pour orienter l'ordre de traitement des nœuds. Cette approche est explorée pour la première fois par P. Hart, N. Nilsson et B. Raphael en 1968 dans [HaNi1968].

L'algorithme A* reprend le principe de l'algorithme de Dijsktra. On dit d'un nœud qu'il est « fermé » si on connait le plus court chemin depuis P vers celui-ci (ensemble noté N_A dans la description de l'algorithme de Dijkstra). On dit d'un nœud qu'il est « ouvert » s'il n'est pas fermé et est un successeur direct d'un nœud fermé (ensemble N_B dans la description de l'algorithme de Dijkstra). « Ouvrir » un nœud signifie donc transférer un nœud dans l'ensemble N_B. « Fermer » un nœud signifie donc transférer un nœud dans l'ensemble N_A.

Supposons que l'on ait défini une fonction d'estimation $\hat{d}(R)$ qui à chaque nœud associe une priorité d'expansion, correspondant à une estimation de la distance d'un chemin minimal de P vers Q passant par R. Lors de l'étape de sélection du nœud à fermer, l'algorithme A* préfèrera choisir non pas le nœud à distance minimale de l'origine P, mais plutôt le nœud minimisant la fonction d'estimation $\hat{d}(R)$.

Les étapes de l'algorithme sont alors les suivantes :

1. Ouvrir P.
2. Choisir le nœud ouvert R d'estimation $\hat{d}(R)$ minimale : si c'est Q, terminer l'algorithme.

3. Sinon, fermer R.
4. Ouvrir ses successeurs R_i qui ne sont pas fermés, ou qui le sont mais dont la nouvelle estimation $\hat{d}(R_i)$ est inférieure à celle avec laquelle il a été fermé. Aller à la deuxième étape.

Le choix de la fonction d'estimation est crucial. En effet, certains choix de fonction $\hat{d}$ ne garantissent pas que l'algorithme trouve effectivement le plus court chemin entre P et Q. Un choix classique de fonction d'estimation pour un nœud ouvert R est la somme de la distance calculée de P à R notée $d_c(P, R)$ et de la distance à vol d'oiseau entre R et Q notée $d_v(R, Q)$.

$$\hat{d}(R) = d_c(P, R) + d_v(R, Q)$$

De manière générale, toute fonction d'estimation $\hat{d}$ composée d'une part de $d_c(P, R)$ et d'autre part d'une borne inférieure sur la distance minimale entre R et Q est acceptable pour l'algorithme A*, au sens qu'elle assure à celui-ci de trouver une solution optimale. Remarquons que si cette borne inférieure est 0, l'algorithme A* est exactement l'algorithme de Dijkstra, et plus cette borne inférieure est élevée, plus l'algorithme est rapide en temps de calcul.

La figure 11 illustre les arcs explorés par l'algorithme A* dans un réseau routier.

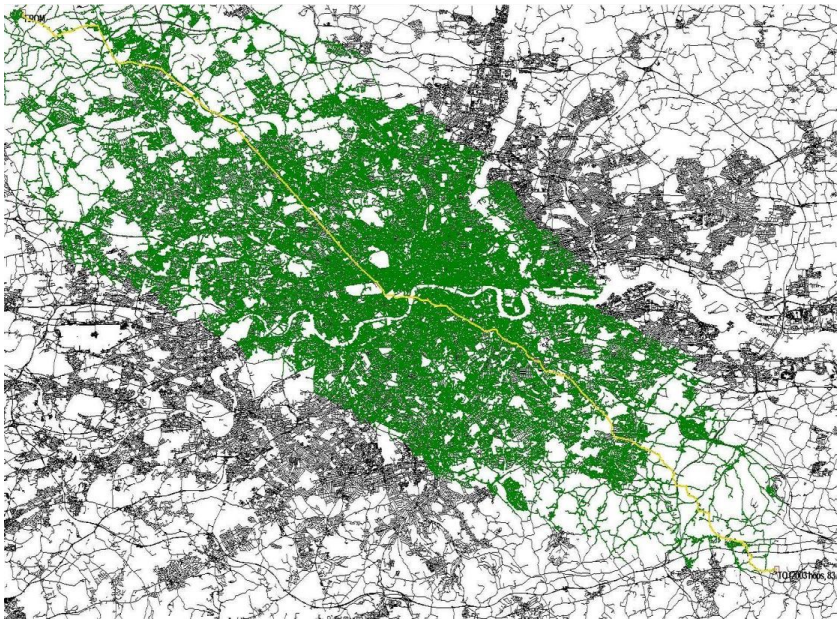


FIGURE 11 – Algorithme A*

3.3 Algorithme ALT

Goldberg et Harrelson introduisent dans [Gold2004] l'algorithme *A* with Triangle Inequality and Landmarks* (ALT). C'est une variante de l'algorithme A* qui effectue un prétraitement pour définir sa fonction d'estimation. L'idée repose sur l'établissement

d'un petit nombre, seize dans leur article, de *Landmarks* (lieux de référence) dans le graphe et l'utilisation de l'inégalité triangulaire pour définir la fonction d'estimation.

Avant tout calcul de plus court chemin, pour chaque lieu de référence L, est calculé le plus court chemin vers et partant de chaque nœud R du graphe, que l'on notera respectivement $d(R, L)$ et $d(L, R)$.

On suppose démontré que ce que l'on appelle distance entre deux nœuds, c'est-à-dire la somme des poids des arcs du plus court chemin les reliant, respecte l'inégalité triangulaire.

Lors du déroulement de l'algorithme A*, pour un nœud R ouvert, les deux inégalités triangulaires $d(R, Q) + d(Q, L) \geq d(R, L)$ et $d(L, R) + d(R, Q) \geq d(L, Q)$ sont vraies pour tout *landmark* L. La fonction b de borne inférieure de la distance minimum $d(R, Q)$ est alors définie par la formule suivante, avec $\mathcal{L}$ un sous-ensemble de *landmarks*, bien choisi en fonction de P et Q.

$$b(R) = \max_{L \in \mathcal{L}} \{d(L, Q) - d(L, R), d(R, L) - d(Q, L)\}$$

Cette formule diminue considérablement l'ensemble de nœuds explorés, et le calcul de la borne dépend du nombre de *landmarks* utilisés. Il faut également noter que plus il y a de *landmarks*, plus l'espace mémoire stockant les matrices de distances est important.

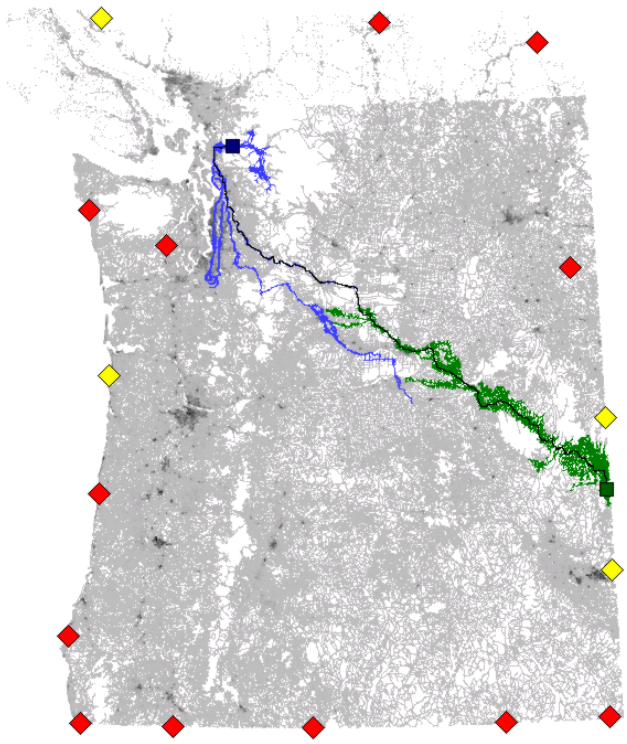


FIGURE 12 – ALT bidirectionnel (Goldberg et al.)

La figure 12 montre les arcs et nœuds explorés par cet algorithme ainsi que la position des *landmarks* utilisés pour calculer la borne inférieure.

3.4 *Arc flags*

La méthode de marquage des arcs (*arc flags*, ou *edge labels*) est une autre technique d'optimisation permettant de réduire le temps de calcul en orientant la direction des arcs explorés par l'algorithme. Elle nécessite un prétraitement qui associe à chaque arc $\mathtt{a}$ du graphe un sur-ensemble de tous les nœuds présents sur un plus court chemin commençant par $\mathtt{a}$. La première application de cette idée, introduite dans [ScWa1999], consiste pré-calculer tous les plus courts chemins en partant de chaque arc $\mathtt{a}$, sans les stocker car cela prendrait trop de mémoire, et conserver une donnée angulaire déterminant $\mathsf{M}(\mathtt{a})$. Une autre approche plus efficace, introduite dans [Laut2004], consiste à découper le graphe en régions et à marquer chaque arc par une ou plusieurs de ces régions.

Pendant le calcul, lorsqu'un nœud R doit être ouvert, il est ignoré s'il n'est pas marqué avec la région de la destination Q.

FIGURE 13 – *Arc flags* [ScWa1999]

La figure 13 montre, dans un réseau de train, à gauche les arcs explorés par l'algorithme Dijkstra, à droite les arcs explorés avec l'optimisation de marquage des arcs.

3.5 *Reach* et *shortcuts*

Gutman propose dans [Gutm2004] une approche d'optimisation basée sur le concept de portée ou « *reach* » des nœuds. Intuitivement, Gutman définit la portée d'un nœud

comme une représentation de la longueur des plus courts chemins qui passent par ce nœud. Un nœud qui a une grande portée est un nœud qui se trouve sur un grand plus court chemin, au sens de la somme des poids qui le composent, ou dans une autre métrique bien choisie. Ainsi, les nœuds sur des autoroutes et des routes nationales ont une plus grande portée que les nœuds sur des routes secondaires.

Dans l'algorithme de Dijkstra modifié, lorsqu'un nœud R doit être ouvert, il est ignoré si la portée du nœud est à la fois inférieure à la distance calculée depuis P et à une borne inférieure de la distance jusqu'à Q du type distance à vol d'oiseau. En effet, si la portée est plus petite, il est impossible qu'un plus court chemin reliant P à Q passe par R.

Le principal défaut de cet algorithme est que le temps de prétraitement est extrêmement élevé : il faut calculer tous les plus courts chemins du graphe. Certaines heuristiques peuvent être effectuées pendant le prétraitement, pour réduire le temps de calcul.

Shortcuts La méthode *Reach* est considérablement améliorée par Goldberg, Kaplan et Werneck dans [Gold2006] grâce à l'intégration de *shortcuts* (arcs de « raccourcis »), représentant un plus court chemin entre deux nœuds reliés par une succession d'arcs. Cela a pour effet de diminuer les valeurs de portée des nœuds intermédiaires. Ainsi, les auteurs observent une accélération à la fois de l'algorithme de prétraitement qui génère les portées de chaque nœud, et de l'algorithme de calcul de plus court chemin lui-même.

La figure 14 illustre les arcs explorés par cet algorithme dans sa variante bidirectionnelle, sans les *shortcuts* à gauche, et avec *shortcuts* à droite.

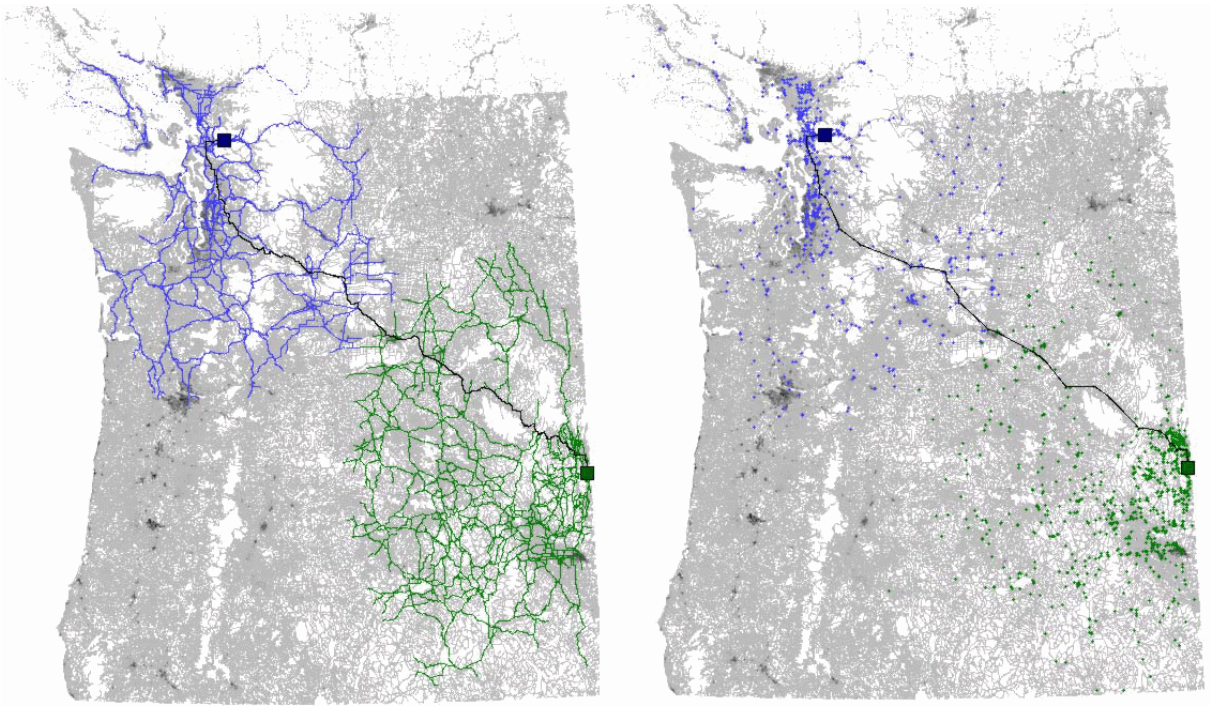


FIGURE 14 – *Reach* / *Reach+Shortcuts* (Goldberg et al.)

3.6 *Contraction Hierarchies*

Le fait que les réseaux routiers sont de nature hiérarchique, composés de tronçons plus importants que d'autres, et l'idée de créer des arcs de « raccourcis » donnent naissance à des techniques d'accélération hiérarchique dont le principe est de trier les tronçons par importance et parcourir le sous-graphe des artères principales.

Les premiers algorithmes utilisant ces idées sont l'« *Highway Node Routing* » introduit dans [ScWa1999], et aussi l'« *Highway Hierarchies* » décrit dans [Sand2006]. Une version améliorée de ces derniers, que nous avons choisi de présenter ci-dessous, est l'algorithme « *Contraction Hierarchies* », proposé par R. Geisberger dans son mémoire [Geis2008].

L'idée est d'ordonner les nœuds du graphe par un ordre total d'« importance », puis de contracter le graphe en ajoutant de manière itérative des arcs de « raccourcis » autour de tous les nœuds du graphe, que l'on supprime virtuellement par ordre d'importance croissante. Cela créé autant de couches superposées de graphes qu'il y a de nœuds dans le graphe, avec le graphe inférieur égal au graphe originel, et les graphes des couches supérieures comprenant de moins en moins de nœuds et d'arcs. Lorsque l'on contracte un nœud u du graphe, on préserve les plus courts chemins du graphe de la couche supérieure en remplaçant tous les chemins de la forme $\langle v, u, w\rangle$ par un arc raccourci de même coût (v, w), seulement si $\langle v, u, w\rangle$ est le seul plus court chemin entre v et w.

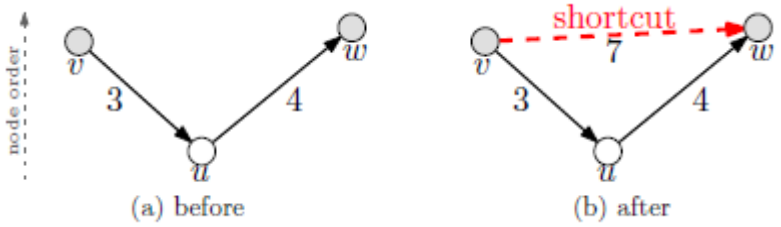


FIGURE 15 – Contraction du nœud u ([Geis2008])

Pour trouver le plus court chemin entre deux nœuds P et Q, on applique au graphe contracté un algorithme de Dijkstra bidirectionnel, qui n'évalue que les arcs menant à des nœuds d'importance plus élevée que leur origine. Pour restituer le résultat, il suffit de dé-contracter récursivement les arcs « raccourcis » du résultat.

L'importance d'un nœud est définie de manière à minimiser le nombre de raccourcis (v, w) ajoutés tout en maximisant le nombre de chemins $\langle v, u, w\rangle$ qu'ils permettent de court-circuiter. Il est également important de contracter le graphe de manière « uniforme ». Ces deux composantes rentrent en compte lors du choix du prochain nœud à contracter.

Nous ne détaillerons pas plus les nombreuses heuristiques et techniques d'optimisations, aussi bien au niveau du calcul de l'importance des nœuds que de l'application des contractions ou des calculs d'itinéraire.

Les temps de prétraitement et les temps de calcul d'itinéraire sont exceptionnellement rapides. En effet, on trouve dans [BDGM2016] des résultats d'expérimentations

sur l'Europe entière qui rapportent un temps de prétraitement de cinq minutes et un temps de calcul d'itinéraire moyen inférieur à une milliseconde, contre plus de deux secondes pour l'algorithme de Dijkstra classique.

3.7 *Hub Labeling*

Parmi les techniques d'optimisation, certaines consistent à n'utiliser que des plus courts chemins déjà calculés lors d'un prétraitement, et pas le graphe lui-même. Parmi celles-ci, nous retiendrons et expliquerons brièvement les algorithmes de *Hub Labeling* ([Abra2011]) et de *Transit Node Routing* ([Bast2006]). Le prétraitement peut lui-même utiliser une des techniques d'optimisation expliquée précédemment, par exemple celle des *Contraction Hierarchies*.

Pour chaque nœud U du graphe, un ensemble de labels $L(U)$, c'est-à-dire de plus courts chemins vers d'autres nœuds, est calculé de façon à ce que pour tout nœud V, il existe un nœud dans $L(U) \cap L(V)$ également inclus dans le plus court chemin de U vers V.

Alors, le plus court chemin entre U et V peut être directement consulté en mémoire en combinant les labels de $L(U)$ avec les labels de $L(V)$, et en sélectionnant le plus court. Nous n'expliquerons pas ici comment choisir les « *hubs* », c'est-à-dire les nœuds intermédiaires entre chaque nœud du graphe.

Cette méthode a pour avantage un temps de calcul quasi-instantané, inférieur à une microseconde, mais elle souffre d'un temps de prétraitement extrêmement long, accompagné d'un espace mémoire requis pour stocker les labels très élevé.

3.8 *Transit Node Routing*

Lorsque un automobiliste part de chez lui pour faire un trajet longue distance, il se retrouve souvent sur une entrée d'autoroute proche. Cette remarque a donné naissance à la technique d'optimisation de *Transit Node Routing*, qui consiste à choisir un petit ensemble de nœuds de transit, de manière à ce que tout plus court chemin de U vers V « suffisamment long » passe nécessairement par un nœud de transit proche de U puis un nœud de transit proche de V.

Le prétraitement consiste à calculer les plus courts chemins de chaque nœud du graphe vers les quelques nœuds de transit proches d'eux à vol d'oiseau, ainsi que tous les plus courts chemins des nœuds de transit entre eux. Ce precalcul est donc légèrement plus rapide, et prend nettement moins de place en mémoire, que celui du *Hub Labeling*. Puis, lors du calcul du plus court chemin de P vers Q, il faut consulter en mémoire et composer les trois chemins suivants : de P vers les nœuds de transit proches de P, des nœuds de transit proches de P vers les nœuds de transit proche de Q, et de ces derniers vers Q. Notons que pour les requêtes courte distance, c'est-à-dire les cas où P et Q sont proches géographiquement, cet algorithme n'apporte aucune amélioration.

3.9 Combinaisons de méthodes d'optimisation

Ces méthodes d'optimisation peuvent être mélangées et donner naissance à de nouvelles méthodes plus performantes, comme par exemple la méthode CHASE, pour *Contraction Hierarchies + Arc Flags*, ou la méthode SHARC, pour *Shortcuts + Arc Flags* ([Dell2009]).

4 Algorithmes de résolution dans un réseau de transport en commun

Comme exposé dans la section 3.2, il existe deux modèles principaux pour la résolution du plus court chemin dans un réseau de transport en commun : les modèles *time-dependant* et *time-expanded*. Nous décrivons ici des algorithmes de résolution sur ces deux modèles, et aussi d'autres algorithmes, comme RAPTOR et *Connection Scan Algorithm*, qui n'exploitent pas de graphe.

4.1 Dijkstra sur les modèles *time-dependant* et *time-expanded*

Si l'on peut appliquer un algorithme de Dijkstra directement sur le modèle *time-expanded*, il n'est pas certain *a priori* qu'il fonctionne dans le cas du modèle *time-dependant*. En effet, pour chaque arc, le poids dépend d'une ressource « horaire actuel » à son nœud d'origine. Alors, il est nécessaire pour que l'algorithme donne un résultat optimal que les fonctions d'évaluation de poids des arcs respectent la contrainte FIFO, c'est-à-dire qu'il ne faut pas qu'un départ ultérieur à un nœud puisse mener à une arrivée antérieure au nœud suivant.

Dijkstra *time-dependant* Pour répondre à la requête $\mathsf{A}@\tau \rightarrow \mathsf{B}$ dans un graphe *time-dependant*, il suffit d'appliquer l'algorithme de Dijkstra avec comme nœud de départ A_t, le nœud de transfert de A, avec un « horaire actuel » égal à τ, et comme nœud d'arrivée B_t, le nœud de transfert de B.

Dijkstra *time-expanded* Pour répondre à la requête $\mathsf{A}@\tau \rightarrow \mathsf{B}$ dans un graphe *time-expanded*, il suffit d'appliquer l'algorithme de Dijkstra avec comme nœud de départ le nœud $\mathsf{A}_t@\tau_i$ avec τ_i la plus petite valeur possible supérieure ou égale à τ, et comme ensemble de nœuds d'arrivée tous les nœuds s'écrivant $\mathsf{B}_t@\upsilon$, avec $\upsilon \geq \tau$.

On trouve dans [Pyrg2008] une analyse poussée de l'application de Dijkstra sur les deux modèles pour les problèmes d'**arrivée au plus tôt** et de **moins de changements**, ainsi que leur combinaison lexicographique et bicritère. L'article est également accompagné des temps d'exécution sur chaque modèle : la modélisation *time-dependant* est clairement plus adaptée. En effet, le graphe est de taille bien moindre, et les arcs représentant le même chemin physique à des horaires différents peuvent être tous évalués

dans le modèle *time-expanded* (alors qu'il n'y a qu'un seul arc dans le modèle *time-dependant*, évalué une seule fois). En revanche, les évaluations des fonctions de poids, en particulier la recherche de l'horaire du prochain départ, pénalisent légèrement les temps de calcul dans le modèle *time-dependant*.

4.2 Optimisations possibles sur les modèles *time-dependant* et *time-expanded*

Nous traitons ici des améliorations proposées par la littérature sur les algorithmes de recherche d'itinéraire dans les graphes TC, inspirées par les méthodes s'appliquant aux graphes individuels.

Recherche bidirectionnelle Une recherche bidirectionnelle est difficile ou impossible dans les deux types de graphes : en effet, on ne connait pas l'horaire d'arrivée à la destination. Pour le graphe *time-dependant*, on ne sait donc pas établir « l'horaire actuel » au nœud d'arrivée, ce qui rend les évaluation de poids des arcs impossible dans le sens inverse. Pour le graphe *time-expanded*, on ne sait pas précisément quel est le nœud d'arrivée optimal, puisqu'on ne connait pas l'horaire d'arrivée optimal. Il faut donc lancer un calcul inverse avec tous les nœuds d'arrivée, ce qui n'apporte pas une accélération du temps de calcul remarquable (voir [Bast2009]).

Hiérarchies et contractions L'expérimentation sur un modèle *time-expanded* a montré que l'utilisation de techniques hiérarchiques, avec la portée ou les contraction de nœuds par exemple, est largement moins efficace que sur des réseaux routiers ([BaDW2007] et [Bast2009]). La raison avancée par Bast est intuitive : les réseaux de bus et de métro ne sont pas hiérarchiques ; pour une requête aléatoire, il n'en existe pas vraiment un plus important que les autres, et le choix d'une ligne plutôt qu'une autre est basé plutôt sur les lieux de départ, d'arrivée et la compatibilité de l'horaire de départ avec les tables horaires.

De plus, le prétraitement est nettement plus lent que sur les réseaux routiers, en particulier avec des réseaux urbains très denses comme ceux de la ville de New-York. Sur des réseaux plus importants, on peut commencer à distinguer une hiérarchie avec des lignes longues distances de train ou de car, permettant de relier plusieurs agglomérations. Mais cela n'empêche pas la difficulté liée aux recherches locales de ralentir le prétraitement.

A* L'algorithme A* peut être appliqué sur les deux modèles de graphe, en prenant comme borne inférieure la distance à vol d'oiseau vers l'objectif divisée par la vitesse du véhicule le plus rapide des modes de transport du réseau considéré. Cependant ce n'est pas une méthode très efficace dans le cas des transports en commun. Cela s'explique par la structure du réseau : l'algorithme A* évite tout chemin s'éloignant du but, or il n'est pas rare qu'un chemin optimal rejoigne un arrêt ou une gare bien desservie à l'opposé de la direction de l'objectif. Pour les réseaux urbains, Bast obtient une accélération des temps CPU d'un facteur deux ou trois dans [Bast2009]. Pour des réseaux incluant

des modes de transport à grande vitesse, comme des trains, cela n'est pas efficace. Les expérimentations de Schulz ([Schu2005]) montrent que l'optimisation de type A* qu'il a développée prend en moyenne plus de temps processeur, même si le nombre de nœuds parcourus est plus faible. Le coût en temps CPU des fonctions nécessaires pour garantir l'optimalité des solutions est supérieur au gain dû à la réduction du nombre de labels.

Notons que l'on trouve dans [Diss2008] une autre fonction pour la borne inférieure, qui nécessite un calcul de plus court chemin préliminaire à partir de l'arrêt d'arrivée B, sans horaire spécifié, et qui utilise comme poids des arcs une borne inférieure des temps de parcours possibles à chaque horaire. Les auteurs obtiennent ainsi une amélioration des temps de calcul d'environ 40 %.

ALT et *Arc Flags* Les méthodes ALT et *Arc Flags* s'appliquent facilement au graphe *time-expanded*, et peuvent se révéler efficaces pour réduire les temps de calcul. Cependant elles nécessitent des temps de prétraitement très importants. En effet, calculer tous les plus courts chemins à partir de certains landmarks (ALT), ou pour aller dans certaines régions du graphe (*Arc Flags*), à tous les horaires de la journée, est nécessairement extrêmement lent.

Delling apporte une réponse à ce problème dans [Dell2009a] en utilisant un graphe simplifié des arrêts, contenant un nœud par arrêt et un arc reliant deux arrêts s'il existe une étape élémentaire entre les deux. Le poids des arcs est la borne inférieure de la fonction de poids dépendant du temps. D'autres optimisations liées à la nature du graphe *time-expanded* permettent de réduire encore la durée du calcul, pour finir par obtenir un temps de calcul moyen d'environ 10ms sur les chemins de fer d'Europe, contre 500ms pour Dijkstra. En revanche, les temps de précalcul restent excessivement longs (plus de quatre jours).

4.3 Dijkstra multilabel sur le modèle par arrêt

Le modèle *time-dependant*, avec un seul nœud par arrêt, est utilisé par Geisberger dans [Geis2010], sur lequel il applique une variante multilabel de Dijkstra. Chaque arc entre deux arrêts est représenté par un ensemble d'étapes élémentaires permettant d'aller du premier arrêt à l'autre. La démultiplication des nœuds par itinéraire du modèle classique est remplacée par la démultiplication des labels à un nœud lors de l'algorithme, avec une règle de dominance dépendant des critères. L'extension d'un label est rendue plus complexe par le fait que les étapes élémentaires d'un arc ne respectent pas nécessairement la règle FIFO.

Pour une requête d'**arrivée au plus tôt**, un label P domine un label Q si :

- P et Q arrivent au même arrêt,
- l'horaire d'arrivée de P est antérieur ou égal à celui de Q,
- P permet d'effectuer au moins les même transferts que Q à leur arrêt d'arrivée.

Nous ne détaillerons pas ici l'algorithme multilabel, parce qu'il est similaire à celui que nous présentons dans le chapitre IV (voir aussi algorithme de Martins, [Mart1984]).

C'est un cas intéressant d'utilisation d'algorithme multilabel pour répondre à une requête monocritère dans un graphe qui ne permet pas une simple application de l'algorithme de Dijkstra. Notons que les résultats d'expérimentations montrent une diminution des temps de calcul, en moyenne cinq fois moins grands qu'une application de l'algorithme de Dijkstra sur le graphe *time-dependant*, sur les réseaux de trains d'Europe, de transport de Berlin-Brandebourg (VBB) et aussi de transport du Rhin (RMV).

4.4 *Contraction Hierarchies* sur le modèle par arrêt

Dans la continuité de son travail sur le modèle par arrêt [Geis2010], Geisberger propose d'appliquer l'algorithme de *Contraction Hierarchies* sur des réseaux de transports en commun. La création même du graphe avec un nœud par arrêt est motivée par le fait que les résultats d'une application brutale de la méthode sur le graphe *time-dependant* ou le graphe *time-expanded* sont décevants. Des travaux ultérieurs de Wirth [Wirt2015] améliorent son approche.

Dans les deux cas, le prétraitement repose sur une élimination progressive des nœuds du graphe par la création d'arcs de « raccourcis », voir à ce propos notre description de l'algorithme de *Contraction Hierarchies*. Cependant, chaque arc du graphe est lié à un ensemble d'**étapes élémentaires**. Lors de l'élimination virtuelle d'un nœud, ces étapes élémentaires sont fusionnées en étapes non élémentaires, ou chemins, qui permettent de relier les nœuds adjacents au nœud à éliminer. Pour cela, une relation de dominance entre chemins de même origine et destination est définie afin de ne pas faire exploser l'ensemble des chemins possibles lié à un arc « raccourci ». La figure 16 nous montre un exemple de contraction de nœud avec fusion des chemins possibles. Il est possible d'introduire les correspondances entre arrêts comme étapes élémentaires, mais il est nécessaire qu'elles respectent une règle de transitivité (A en correspondance avec B et B avec C implique que A l'est avec C), sans quoi l'algorithme générerait de nouvelles correspondances (*a priori* interdites).

L'ordre d'élimination des nœuds se définit avec le même objectif que la méthode originale, à savoir réduire le nombre d'arc générés et augmenter le nombre d'arcs court-circuités.

Il est important de noter qu'une recherche inverse partant de l'arrêt d'arrivée de la requête, nécessaire à l'algorithme des *Contraction Hierarchies*, est impossible dans un graphe *time-dependant*, comme l'est celui-ci. Pour pallier ce problème, chaque arc « descendant », c'est-à-dire dans le sens inverse de la hiérarchie des nœuds, susceptible de mener à l'arrêt d'arrivée est marqué. Le calcul direct est une application du Dijkstra multilabel sur le modèle par arrêt, seuls les arcs montant et les arcs descendants marqués sont explorés.

De très nombreuses optimisations et règles supplémentaires ont été développées dans [Geis2010] et [Wirt2015] pour accélérer les processus de prétraitement et de calcul d'itinéraire. Afin d'alléger la lecture de ce manuscrit, nous ne les détaillerons pas ici.

Les temps de prétraitement de ces algorithmes sont relativement bas, de l'ordre de la minute pour les réseaux naturellement hiérarchisées comme les trains d'Europe. Ils dépassent néanmoins les quinze minutes pour les réseaux de transport urbains denses

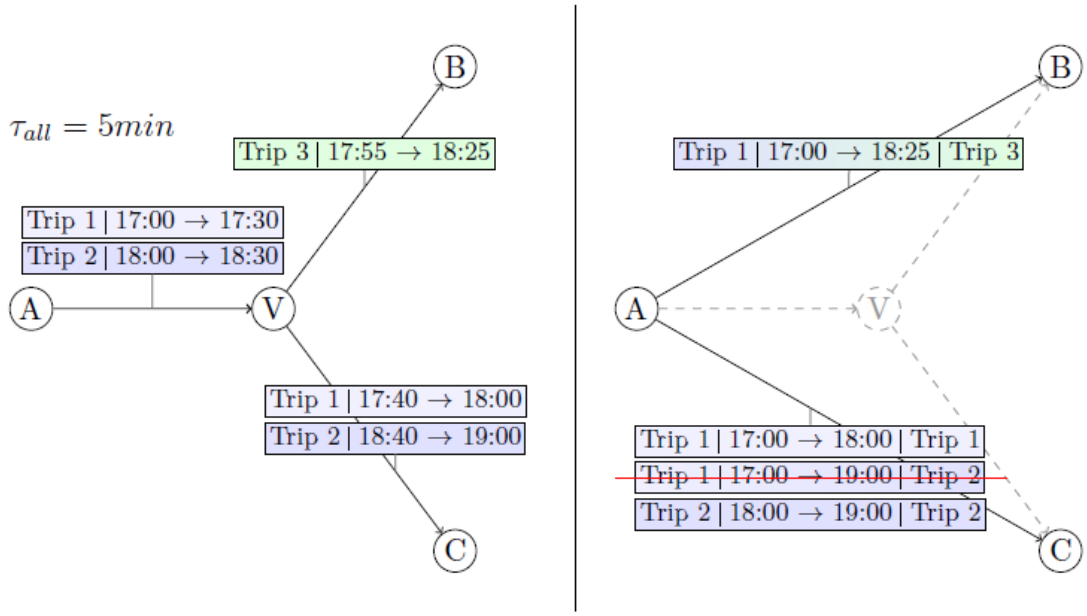


FIGURE 16 – Contraction du nœud V dans le graphe par arrêt ([Wirt2015]) et combinaison des chemins passant par V. Le chemin partant de A à 17:00 et arrivant à C a 19:00 est éliminé

comme ceux de la ville de Chicago. Les temps de calcul sont quant à eux deux à quarante fois plus rapide que sur le graphe par arrêt non contracté, dix à six-cent fois plus rapide que l'algorithme de Dijkstra sur le graphe *time-dependant* classique, amenant souvent la moyenne du temps de réponse en dessous de la milliseconde. Ces résultats sont basés sur [Wirt2015], qui a réalisé ses expérimentations sur de nombreux réseaux de transport en commun.

4.5 *Transfer Pattern Routing*

Dans le même travail [Geis2010], Geisberger introduit l'une des techniques d'optimisations les plus efficaces connues à ce jour pour calculer des itinéraires en transport en commun, le *Transfer Pattern Routing*. Qui plus est, cette méthode s'adapte facilement au cas multicritère et à la possibilité de rabattement en marche à pied, vélo et voiture. Son principal inconvénient est un temps de prétraitement et un espace de stockage des données prétraitées extrêmement élevés.

Geisberger appelle *Transfer Pattern* d'un trajet réalisable la suite des arrêts de montée et descente de ce trajet, c'est-à-dire les arrêts de départ et d'arrivée, ainsi que les arrêts intermédiaires de correspondances. Nous traduirons cette notion par schéma de transfert.

L'idée principale de cette méthode repose en partie sur l'établissement au préalable de tous les schémas de transfert optimaux pour chaque couple d'arrêts départ/arrivée du réseau, pour tout horaire de la journée et pour tout type de journée. D'autre part, il faut avoir une structure de données permettant pour tout couple d'arrêts et horaire

de trouver les horaires de la prochaine course directe reliant ces arrêts.

Ainsi, lorsque nous recherchons le trajet d'**arrivée au plus tôt** entre les arrêts A et B, il suffit de prendre chaque schéma de transfert $A = C_1, \ldots, C_n = B$ lié au couple (A, B) et d'appliquer la recherche de course directe pour chaque couple successif C_i, C_{i+1}. Cette dernière est très rapide (de l'ordre de dix microsecondes), et le nombre de schéma optimaux n'est pas très élevé, les calculs sont donc extrêmement rapides. L'ensemble de trajets obtenu est présenté tel quel ou filtré pour ne conserver que les résultats les plus intéressants.

En réalité, il est trop long et trop couteux de stocker les schémas de transfert pour tous les couples d'arrêts dans certains réseaux. En effet, le nombre d'arrêts dans un réseau urbain peut atteindre plusieurs dizaines de milliers, ce qui entraine plus d'un milliard de couples possibles. Pour pallier ce problème, Geisberger propose de ne pas stocker tous les schémas de transfert possible, mais un sous-ensemble de ceux-ci, tel que l'on puisse recombiner à partir des schémas stockés les schémas manquants. En se basant sur une sélection de « *hubs* » choisis en fonction de leur importance (déterminée par exemple par leur « *reach* » dans le graphe *time-expanded*), la méthode précalcule uniquement les schémas entre *hubs* et tous les autres arrêts du graphe, qu'il appelle « globaux », ainsi que les schémas « locaux » des arrêts du graphes vers les *hubs* à proximité. Même ainsi, obtenir des temps de prétraitement corrects nécessite le rajout de plusieurs heuristiques.

Les temps de calcul de cet algorithme sont similaires à ceux de la méthode des *Contractions Hierarchies* sur le graphe par arrêt, de l'ordre de la milliseconde. En revanche, en fonction des heuristiques employées, le temps de prétraitement peut aller de plusieurs dizaines d'heures jusqu'à plusieurs centaines d'heures pour un réseau urbain dense ([Geis2010], [BDGM2016]). L'avantage par rapport aux *Contractions Hierarchies* est qu'il peut prendre en compte plusieurs critères, ce qui justifie ces temps de prétraitement.

4.6 RAPTOR (*Round-Based Public Transit Routing*)

Dans [Dell2012], Delling, Pajor et Werneck présentent un nouvel algorithme sans file de priorité, sans parcours de graphe, et sans prétraitement, permettant de répondre aux requêtes 1-n d'**arrivée au plus tôt** et de **moins de changements**, ainsi qu'à la combinaison des deux. Cet algorithme est également présenté en détails dans la thèse de doctorat de Pajor ([Pajo2013]).

Cet algorithme est basé sur le parcours des tableaux des courses et itinéraires du réseau. Organisé en « *rounds* », cet algorithme de programmation dynamique considère le problème suivant. Si l'on connait avec certitude les trajets optimaux depuis A en i ou moins de changements, comment calculer les arrêts accessibles en $i+1$ changements. Pour y répondre, il suffit d'examiner tous les itinéraires des arrêts atteints par des trajets optimaux en exactement i changements. Examiner un itinéraire signifie parcourir tous les arrêts traversés par l'itinéraire, dans l'ordre, en maintenant à jour la course qu'il est possible d'emprunter au plus tôt tenant compte des arrêts atteints au *round* précédent. Il suffit d'initialiser l'algorithme avec A accessible en 0 changements, et répéter les

rounds jusqu'à ce qu'aucune amélioration de trajet optimal ne soit trouvée.

Les correspondances marche à pied peuvent être traitées, à condition qu'elle soient transitivement fermées. Il suffit de propager entre chaque *round* tout nouveau trajet optimal vers les arrêts en correspondance avec le dernier arrêt atteint.

De fait, l'algorithme calcule pour chaque arrêt la liste de trajets suivante : le trajet optimal sans changement, le trajet optimal en maximum un changement, le trajet optimal en maximum deux changements, etc.

Il existe une variante multicritère de l'algorithme, McRAPTOR, qui consiste à conserver non plus le trajet le plus rapide à chaque *round*, mais le ou les trajets non dominés sur critères spécifiés.

Vu qu'il y a relativement peu d'itinéraires, qu'il n'y a pas de file de priorité à dépiler, et que le nombre de *rounds* est souvent très faible, le temps de calcul de cet algorithme est en moyenne deux fois inférieur à l'application de l'algorithme de Dijkstra sur le graphe *time-dependant*. De plus, il génère plus de solutions puisqu'il répond au problème bicritère « **arrivée au plus tôt**, **moins de changements** ».

4.7 Tri topologique et *Connection Scan Algorithm*

Vu que le graphe *time-expanded* est acyclique, il est possible de trier ses nœuds dans un ordre topologique, et d'explorer le graphe dans cet ordre. Chaque nœud est visité une seule fois de manière à ce qu'un sommet soit toujours visité avant ses successeurs. Cela réduit la complexité théorique de l'exploration, par rapport à l'utilisation d'une file de priorité.

L'article [Dibb2013] introduit un algorithme basé sur ce principe, nommé « *Connection Scan Algorithm* ». Comme son nom l'indique, il s'agit de parcourir toutes les étapes élémentaires de transport en commun dans l'ordre des horaires de départ, ce qui revient à parcourir les arcs *time-expanded* dans un ordre topologique. Les arrêts, les correspondances marche à pied et les étapes élémentaires du réseau de transport en commun sont implémentés comme tableaux de données, et non comme un graphe. Chaque étape élémentaire inclut également une référence à la suivante et à la précédente dans la même course, ainsi qu'un retard éventuel maximal de la course.

L'ensemble des étapes élémentaires est inséré dans un tableau trié par horaire de départ, accompagné de pointeurs vers la course et l'itinéraire correspondant, comme le montre le tableau 4.

L'algorithme maintient alors pour chaque arrêt un label d'**arrivée au plus tôt**. Ces labels sont initialisés à l'infini sauf pour l'arrêt de départ, et mis à jour à chaque évaluation d'étape élémentaire de transport. La prise en compte du temps de transfert minimum à l'arrêt est possible, moyennant la mise en place de marqueurs de dernière course empruntée. La prise en compte de la marche à pied nécessite de rajouter une étape à l'algorithme, qui traite les arrêts en correspondance à chaque fois qu'un label est généré ou amélioré. La prise en compte du multicritère consiste à transformer l'algorithme pour qu'il maintienne plusieurs labels non dominés par arrêt. Cette méthode a été améliorée dans [Stra2014] par l'ajout d'une phase de prétraitement basée sur un partitionnement à

Arrêt départ	horaire	Arrêt arrivée	horaire	Course	Itinéraire
Gare	7:00	École	7:05	C4	I2
École	7:05	Mairie	7:25	C4	I2
Gare de Colmar	7:50	Gare de Strasbourg	8:50	C6	I4
Mairie	8:00	École	8:15	C1	I1
École	8:15	Gare	8:25	C1	I1
Gare de Strasbourg	9:30	Gare de Colmar	10:30	C8	I5
Mairie	12:00	Gare	12:18	C2	I3
Gare de Colmar	15:30	Gare de Strasbourg	16:30	C7	I4
Gare	16:30	École	16:34	C5	I2
École	16:34	Mairie	16:50	C5	I2
Gare de Strasbourg	17:35	Gare de Colmar	18:35	C9	I5
Mairie	18:00	École	18:15	C3	I1
École	18:15	Gare	18:25	C3	I1

Tableau 4 – Exemple de modélisation CSA

plusieurs niveaux des arrêts, et un traitement différent pour les trajets de type « longue distance ». Au final les performances observées sont meilleures que celles de RAPTOR, et les temps de prétraitement dépendent de la taille de l'instance considérée.

Chapitre III

Recherche d'itinéraire chez Cityway

Sommaire

Ce chapitre a pour objectif de décrire comment le calculateur d'itinéraire de Cityway s'intègre dans l'état de l'art : quelles modélisations et quels algorithmes sont utilisés pour la résolution, quels types de requêtes sont résolues et sous quelles contraintes. Nous nous attacherons également à exposer précisément l'ensemble des contraintes industrielles qui se présentent. Enfin, nous comparerons cela avec l'état de l'art, et discuterons des pistes d'amélioration que nous avons traitées pour nos travaux de thèse.

Le chapitre est découpé en quatre sections : la section 1 présente les modèles et algorithmes liés aux réseaux de transport en commun, la section 2 est consacrée aux réseaux individuels, la section 3 explique comment le calculateur combine les réseaux. Enfin, la section 4 expose les limitations de l'approche de Cityway, et les axes de recherche que nous avons choisi d'explorer.

1 Transport en commun

Les données de transport en commun proviennent directement des transporteurs, dans des formats divers et variés. On peut citer le GTFS, format libre créé par Google, plus répandu dans les pays anglo-saxons, mais aussi les standards Transxchange, NEPTUNE, NeTEX, SIRI, IFOPT, Transmodel et d'autres formats non standardisés. L'intégration et le traitement de ces données est hors du cadre de la thèse, mais il est important de noter que la qualité des données est intimement liée au bon fonctionnement du calculateur. De plus, leur complexité et leur exhaustivité déterminent en partie celles du calculateur.

Ces données sont converties dans un format propriétaire conçu par Citway, et sto-

ckées en base de données. Les tables de la base de données fournissent les informations liées aux lignes, arrêts logiques, arrêts physiques, itinéraires, courses et horaires nécessaires au calcul d'itinéraire. Ces données sont utilisées pour générer les graphes de transport en commun.

1.1 Modèle de graphe

La modélisation des réseaux de transports en commun est un graphe *time-dependant*. Chaque arrêt physique du réseau est représenté par plusieurs nœuds du graphe TC : exactement un nœud de marche (nœud MARCHE) et un ou plusieurs nœuds d'arrêt sur itinéraire (nœud ITI). A chaque itinéraire ne correspond qu'un seul mode de transport, nous pouvons donc associer à chaque nœud le mode de l'itinéraire qui le traverse.

Comme l'illustre la figure 17, il existe dans le graphe TC quatre type d'arcs.

- arc ITI : ce type d'arc relie deux arrêts qui se succèdent directement dans un **itinéraire**, toujours entre deux nœuds ITI.
- arc CORRESPWALK : ce type d'arc relie deux arrêts qui sont en **correspondance marche à pied**, toujours entre leurs nœuds MARCHE.
- arc GETIN : ce type d'arc représente une **montée** dans un transport en commun et relie toujours le nœud MARCHE d'un arrêt avec un nœud ITI.
- arc GETOFF : ce type d'arc représente une **descente** d'un transport en commun et relie toujours un nœud ITI avec son nœud MARCHE.

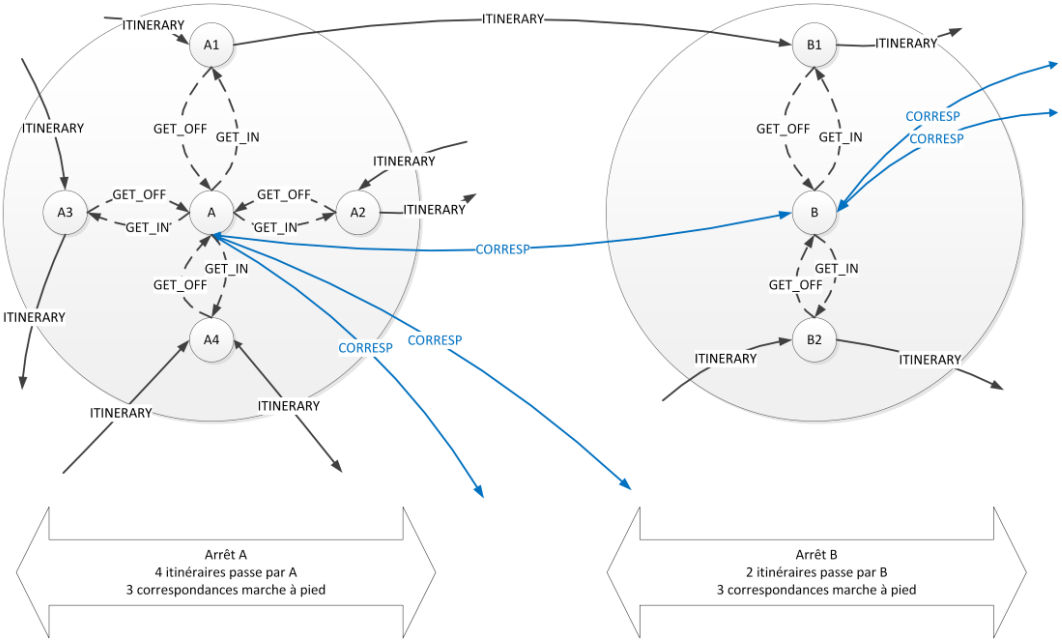


FIGURE 17 – Modèle du graphe TC illustré sur deux arrêts

Cette modélisation est identique à celle du graphe *time-dependant* présentée dans la section 3.2.1 du chapitre I.

Un graphe « graphe *reverse* » est également généré : il s'agit du même graphe que

le modèle précédent, sauf que tous les arcs sont inversés. Les ARCS ITI vont dans le sens inverse de parcours réel du véhicule. Les arcs de montée vont du nœud ITI vers le nœud MARCHE et les arcs de descente du nœud MARCHE vers le nœud ITI. Ce graphe sert à résoudre les requêtes en *arrivée à* selon le critère de **départ au plus tard**. De plus, il sert à résoudre les requêtes en *départ à* selon le critère **lexicographique de durée minimale**. Après que l'algorithme trouve un trajet arrivant au plus tôt, un calcul est systématiquement effectué sur le graphe graphe *reverse* pour s'assurer qu'il est également optimal selon le critère du **départ au plus tard**. Le chapitre VI de cette thèse est consacré à l'étude de ce critère **lexicographique** qui assure d'obtenir des résultats de durée minimale.

Les horaires sont stockés en mémoire comme des tableaux de données, accessibles en temps constant à partir des index de course et d'arrêt. Cette structure permet également les requêtes du prochain départ à un arrêt pour un itinéraire donné, en bouclant sur les courses existantes, jusqu'à ce que l'horaire soit atteint. Notons que la structure accepte les cas où un horaire de départ est différent d'un horaire d'arrivée pour une course à un arrêt. C'est un cas exceptionnel dans les réseaux urbains, mais courant dans les réseaux ferroviaires. Enfin, un fichier horaire est généré par jour de l'année, mais tous les jours ne sont pas chargés en mémoire en même temps.

1.2 Particularités modélisées

Nous allons présenter une liste aussi exhaustive que possible des particularités liées aux données de transport en commun que le calculateur prend en compte, accompagnée d'une brève explication sur la façon dont elles sont intégrées. Dans le reste de la thèse, nous ne reviendrons pas nécessairement sur ces points, mais leur nombre et la complexité qu'ils engendrent est un argument important des choix que nous avons faits.

Arrêts logiques Les points de départ et d'arrivée sont souvent des arrêts logiques, c'est-à-dire un regroupement géographique ou logique de véritables points d'arrêt physiques. Le moteur en a connaissance, et les calculs d'itinéraire d'arrêt à arrêt sont donc en général n-m et pas 1-1.

Courses cycliques Si un itinéraire passe deux fois par le même arrêt, par exemple (A, B, A, D), deux nœuds liés à l'arrêt sont créés. Ici l'arrêt A aura non pas un mais deux nœuds liés à l'itinéraire. Cela assure qu'il n'est pas possible d'éviter l'arrêt B lors du parcours des arcs de l'itinéraire.

Mode porté par les nœuds Normalement, les modes sont déterminés par les courses. Mais il arrive que des courses du même itinéraire aient différents modes, par exemple une ligne de RER remplacée par un CAR à certains horaires. Nous avons fait le choix de dupliquer les nœuds des itinéraires dans ce cas de figure. Ainsi chaque nœud ITI est porteur d'un et un seul mode. Cela permet de comparer les labels de chaque mode, d'interdire et pénaliser les modes sans maintenir plusieurs labels par nœud.

Interdictions de montée ou descente Certains réseaux imposent des restrictions à la montée ou descente de véhicule à certains arrêt. Elles sont gérées par arrêt et

par itinéraire : les arcs de type GETIN ou GETOFF correspondant seront marqués comme interdits à la création du graphe.

Temps réel Certains réseaux apportent en temps réel les localisations et horaires des prochains passages de leurs véhicules. La prise en compte de cela est assez simple dans le modèle actuel : un fichier d'horaires temps réel est généré périodiquement, et est chargé par le calculateur pour modifier les tableaux horaires. Notons que la prise en compte du temps réel dans un modèle *time-expanded* ou avec de lourds prétraitements est nettement plus complexe.

Perturbations Dans un réseau de transport, il n'est pas rare que des arrêts ou des lignes soient fortement perturbées. Il est possible de désactiver pour une période donnée une ligne, un arrêt, un couple ligne/arrêt, ou une course. Il est important de lier chaque arc du graphe à l'ensemble des perturbations concernées avec des accès en $\mathcal{O}(1)$. Une difficulté supplémentaire est également de tenir compte des dates de validité des perturbations.

Horaires de nuit Le calcul n'utilise qu'un fichier horaire, valable pour une journée d'exploitation définie sur une période de 24h. Par conséquent, les transitions d'une journée à l'autre peuvent poser un problème. Pour y pallier, le fichier horaire contient, en sus des horaires de la journée, les horaires de la veille et du lendemain pour couvrir une période plus grande, en général de 32h. Ainsi le même fichier horaire peut contenir la même course en double.

Correspondances multiples L'ensemble des correspondances marche à pied possibles est prédéterminé, et n'est pas nécessairement transitif : il peut y avoir une correspondance entre A et B, et une autre entre B et C, mais aucune entre A et C. Un trajet n'est par ailleurs pas valable s'il emprunte plusieurs correspondances en succession directe. Pour l'éviter, chaque évaluation d'arc CORRESPWALK est rejetée si l'arc évalué immédiatement avant par le label est un arc CORRESPWALK.

Temps de transfert minimums Les temps de transfert minimums P_t sont spécifiés par mode, et augmentent d'autant la durée des arcs de descente et de montée. Par exemple, il est possible de décider que le métro nécessite au moins une minute pour monter et descendre : il faudra donc un intervalle d'au moins deux minutes entre deux métros. Certains arrêts possèdent des temps de transfert spécifiques qui prennent la priorité sur le mode. Chaque arc de montée ou de descente est donc marqué d'un index de mode de transport, afin de garantir l'accès à ces temps supplémentaires en $\mathcal{O}(1)$.

Exceptions aux temps de transfert Les arrêts de départ du calcul ne sont pas affectés par les durées minimums de temps de transfert. En effet, lorsqu'un utilisateur demande un départ à 8h, et qu'il existe une course partant précisément à 8h de l'arrêt de départ spécifié, il est souhaitable que celle-ci puisse apparaître dans les résultats. De plus, il faut que le calcul soit « symétrique », afin que les calculs sur le graphe TC graphe *reverse* soient effectués dans les mêmes conditions que sur le graphe TC direct. Par conséquent, les arrêts d'arrivée aussi ne sont pas affectés par les durées supplémentaires de montée et de descente.

Rester dans le véhicule Des réseaux urbains laissent à leurs utilisateurs la possibilité de rester dans un véhicule au terminus d'une ligne, jusqu'à ce qu'il redémarre par le début d'un autre itinéraire (sens retour, autre ligne, ligne circulaire). Les courses partageant le même véhicule avec possibilité de rester dedans ont le même id de block. Lorsque le temps d'arrêt est inférieur au temps de transfert minimums acceptable, il est nécessaire d'assurer que le calculateur puisse tout de même faire cette étape. De plus, elle ne représente pas vraiment un « changement », au sens du critère du moins de changements. Pour modéliser les blocks, des arcs de type STAY sont créés entre les nœuds ITI d'un même arrêt. Lors de l'évaluation de cet arc, seule une course partageant le même id de block que la dernière course est acceptée.

1.3 Contraintes supplémentaires liées à la requête

L'utilisateur ou le client peut rajouter des contraintes à la requête. Le traitement de celles-ci est similaire à celui des contraintes liées aux données de transport. Par exemple, une interdiction de ligne se traite de la même façon qu'une perturbation d'une ligne : c'est la source de la contrainte qui est différente. Nous allons en donner une liste exhaustive.

Interdictions L'utilisateur peut vouloir calculer ou recalculer un trajet en interdisant au moteur un ou plusieurs opérateurs, modes, lignes ou arrêts. Par exemple, il peut souhaiter obtenir un trajet en métro uniquement. Autre exemple, si un résultat ne lui convient pas à cause de la présence d'une ligne souvent en retard, il peut choisir de relancer le calcul en l'interdisant. Ces interdictions génèrent des marqueurs sur les arcs de montée et descente concernés, marqueurs utilisés lors de leur évaluation pour empêcher l'extension de label.

Attributs de courses Certaines courses peuvent avoir des attributs, comme le vélo à bord, ou l'accueil facilité des personnes en fauteuil roulant. Il est possible de demander au calculateur uniquement des trajets dont chaque étape respecte un ensemble d'attributs. Cette vérification est faite lors de la consultation des prochains horaires, et les courses ne correspondant pas à la requête sont rejetées au profit des suivantes.

Vitesse de marche à pied Le calculateur doit pouvoir moduler la vitesse de marche à pied, ce qui change l'évaluation des arcs de correspondance.

1.4 Algorithme et critères

L'algorithme utilisé est une adaptation de l'algorithme de Dijkstra sur graphe *time-dependant* présenté en chapitre II. Il est caractérisé par l'usage d'un tas binaire qui permet d'extraire le nœud de coût minimum. Il intègre toute la complexité des données, des contraintes et des critères que les clients de Cityway souhaitent prendre en compte. Les contraintes sont prises en compte par le fait que l'extension d'un label n'est pas obligatoire à chaque arc, et peut échouer, pour diverses raisons. Les critères de recherche

sont traités par la modification des poids de certains arcs, sous la forme d'un coût additif ou d'un coefficient multiplicatif.

Temps De manière classique dans l'application de Dijkstra sur un graphe *time-dependant*, la durée d'un arc est déterminée lors de son évaluation, en fonction de l'horaire de départ, et donne son poids initial.

Nombre de changements Limiter le nombre de changements est fait grâce à des pénalités de transfert. Un coût additif $\mathsf{P_c}$ est lié à chaque montée et chaque descente, appliqué lors de l'évaluation de ces arcs. De plus, il est possible de faire un calcul avec comme critère principal le « moins de changements » : cela est géré en augmentant considérablement $\mathsf{P_c}$.

Priorité par mode La calculateur a la particularité de pouvoir renvoyer des trajet qui évitent certains modes, sauf si nécessaire. Afin de privilégier certains modes par rapport à d'autres, le coût additif $\mathsf{P_c}$ de montée et descente dépend du mode du nœud ITI concerné.

Priorité d'arrêts et de lignes Il est possible que les clients souhaitent voir apparaître dans les résultats certains pôles d'échanges, ou certaines lignes, plus ou moins souvent que d'autres. Les poids de montée et descente $\mathsf{P_c}$ à ces arrêts ou lignes sont modifiés par un coefficient multiplicateur paramétrable. C'est une fonctionnalité très rarement utilisée.

Marche à pied Afin de privilégier les correspondances minimisant le temps de marche, un coefficient multiplicateur C_w est appliqué au coût des arcs de type CORRESPWALK. De plus, il est possible de faire un calcul avec comme critère principal le « moins de marche à pied » : cela est géré en augmentant considérablement C_w.

Seuil de marche à pied Certains clients ont demandé à ce qu'il y ait un seuil de durée au dessous duquel le coefficient multiplicateur C_w ne s'applique pas : en effet, marcher dix secondes ou une minute entre deux arrêts n'est pas très différent. Discriminer des trajets sur ce critère est inutile voire contreproductif.

Maximiser le temps d'attente En option, le calculateur peut maximiser les temps d'attente à l'arrêt, afin de laisser à l'utilisateur une marge confortable lors de ses transferts. Si deux trajets partiels comparés ne diffèrent pas en coût, le trajet préféré sera celui qui maximise le dernier temps d'attente, préalablement sauvegardé dans le label.

Le détail précis de l'algorithme figure dans le chapitre IV, consacré à son amélioration.

1.5 Requêtes n-m et isochrones

Il est possible de spécifier plusieurs arrêts de départ, ou plusieurs arrêts d'arrivée, à une requête dans le graphe de transport en commun. Cela est utile pour les trajets combinés, comme décrit en section 3. Il n'est pas question ici de trouver tous les trajets

plus rapides commençant par chacun des points de départ, mais le meilleur trajet parmi toutes les combinaisons possibles d'arrêts de départ et d'arrivée spécifiés.

Arrêts et horaires de départ multiples Chaque nœud MARCHE correspondant à un arrêt de départ est initialisé avec son horaire de départ et un coût correspondant à la différence entre son horaire de départ et l'horaire minimum de tous les horaires de départ. Ces nœuds sont ensuite ajoutés dans la file de priorité à l'initialisation de l'algorithme. Le reste de l'algorithme est identique : les trajets sous-optimaux sont écrasés si des labels provenant d'arrêt de départ distincts se rejoignent à un nœud.

Arrêts d'arrivée multiples Les nœuds MARCHE correspondant aux arrêts d'arrivée sont marqués comme nœuds de fin. La condition d'arrêt de l'algorithme est inchangée : lorsque le coût du nœud sortant de la file de priorité est supérieur à un label de nœud final, on sait que ce label ne peut plus être amélioré. Le label sélectionné pour générer le trajet optimal est bien sûr le label de fin de coût minimal.

Traitement différent des arcs finaux Nous avons vu que les temps de transfert ne s'appliquent pas sur les nœuds d'arrivée. Le fait que l'évaluation d'un arc partant d'un nœud de départ ou arrivant sur un nœud d'arrivée soit différente de l'évaluation normale doit être pris en compte dans le cas d'arrivée multiples. En effet, un label arrivant sur un nœud d'arrivée peut soit être prolongé pour aller vers un autre nœud d'arrivée, soit être considéré comme label de fin du calcul. Pour cette raison, chaque nœud d'arrivée possède potentiellement deux labels plutôt qu'un seul : un label « normal » et un label « de fin ». Lors de l'évaluation d'un arc menant à un nœud marqué comme final, deux évaluations sont faites plutôt qu'une.

Requête 1-n, ou isochrone Il arrive qu'au lieu d'une recherche de trajet optimal entre deux points, il soit demandé l'ensemble des arrêts et horaires d'arrivée de tous les trajets optimaux du (ou des) point(s) de départ spécifiés vers une partie des points d'arrêts du graphe. L'algorithme utilisé est inchangé, seule la sortie change : une liste des nœuds MARCHE atteints par l'algorithme, accompagnée des temps d'accès et du nombre de changements.

2 Transport individuel

Une certaine partie des trajets ne se fait pas sur les réseaux de transport en commun, mais sur les réseaux routiers : chemins, rues, avenues, routes, autoroutes, etc. Les données de Cityway proviennent de fournisseurs cartographiques tels que Navteq [1], OpenStreetMap [2] ou d'autres moins courants. L'importation, la vérification et la transformation de ces données dans la base de données cartographiques de Cityway est hors du cadre de cette thèse.

1. https://here.com/en/navteq
2. http://wiki.openstreetmap.org/wiki/About_OpenStreetMap

Ces réseaux sont modélisés par des graphes similaires à ceux décrit dans la section 3 exposant l'état de l'art en la matière. Pour rappel, chaque arc représente un chemin, ou tronçon de voirie, et chaque nœud une intersection de plusieurs tronçons.

Cityway a choisi de modéliser trois réseaux en trois graphes distincts :

- le graphe *piéton* contenant uniquement les chemins que l'on peut parcourir à pied,
- le graphe *vélo* contenant uniquement les chemins que l'on peut parcourir à vélo,
- et le graphe *routier* contenant uniquement les chemins que l'on peut parcourir en voiture ou autre véhicule motorisé personnel.

À chaque tronçon de voirie sont associés deux arcs orientés, un pour chaque direction. Chaque arc possède une distance en mètres. Il n'y a pas de graphe graphe *reverse* : les requêtes en *arrivée à* se font sur le même graphe, à la différence que les arcs sont parcourus dans le sens inverse, et qu'à chaque nœud ouvert, c'est la liste des arcs entrants dans ce nœud qui est utilisée pour étendre le label.

2.1 Spécificités du calcul sur graphes individuels

L'algorithme utilisé par Cityway est basé sur l'algorithme de Dijkstra, avec quelques spécificités que nous allons lister ici.

Rayon de recherche *Tous les graphes.* Bien souvent, une distance de parcours maximale est précisée à la requête. Vu que distance et durée sont proportionnelles sur les graphes individuels, cela revient à rajouter une condition d'arrêt à l'algorithme de Dijkstra : si un label sort de la file de priorité avec une distance supérieure à la limite, le calcul s'arrête.

Sens interdits *Graphe routier.* Un certain nombre de tronçons de voirie ne peuvent pas se parcourir dans les deux sens. Un seul arc dans la direction correspondant au sens autorisé est créé pour ce type de tronçon.

Vitesses paramétrable *Graphe foot et bike.* La vitesse en marche à pied ou à vélo est paramétrable, le temps mis à parcourir chaque tronçon est donc calculé lors des évaluations des arcs.

Vitesses moyennes *Graphe routier.* La vitesse moyenne d'un véhicule particulier sur un tronçon dépend de la vitesse limite sur le tronçon, et du milieu, urbain ou non. Chaque arc du graphe routier contient donc une information de vitesse.

Arrêts de transport en commun *Tous les graphes.* Afin de pouvoir lier les graphes de transport en commun et les graphes routiers, les arrêts sont intégrés dans le graphe comme des nœuds. Chaque nœud de type arrêt est relié au tronçon le plus proche. Plus de détails sur nos travaux de modélisation à ce propos figurent en annexe A de ce document.

Lieux publics *Tous les graphes.* Les lieux publics sont aussi intégrés au graphe, de la même façon que les arrêts, et permettent de partir ou d'arriver vers ceux-ci. Ils peuvent être de plusieurs types, un exemple notable est la station vélo, qui permet

de combiner des trajets contenant du transport en commun, de la marche, et du vélo entre stations vélo.

Départ et arrivée précis *Tous les graphes.* Une adresse ou un point sur une carte se traduit en couple latitude/longitude. Lorsque l'utilisateur souhaite en partir, le tronçon routier le plus proche est utilisé pour déterminer les arcs de départ du calcul. Un nœud temporaire est rajouté à la volée au graphe, relié aux extrémités des arcs, et est utilisé comme nœud de départ. Cela est particulièrement utile lorsque le tronçon de voirie de départ ou d'arrivée est long, parfois de l'ordre de plusieurs kilomètres, et permet d'avoir un résultat à la fois optimal et proche de la réalité de l'utilisateur.

Vitesses en temps réel et prédictives *Graphe routier.* Un certain nombre de tronçons de voirie, en général les grands axes routiers, peuvent avoir des capteurs de trafic. Ces capteurs donnent une vitesse moyenne en temps réel des tronçons en question. De plus, certains logiciels tiers extrapolent ces données pour prédire les vitesses approximatives sur ces tronçons pour une période d'environ une heure. Le calculateur peut intégrer ces données par l'intermédiaire de fichiers mettant à jour ces données, régulièrement chargés en mémoire. Chaque évaluation sur un tronçon se fait en fonction de l'horaire sur le nœud de départ : le calcul devient **dépendant à l'horaire**. On supposera que la condition FIFO est respectée, bien qu'aucune vérification ne soit faite.

Vitesses historisées *Graphe routier.* Il est possible de collecter les vitesses réelles des tronçons surveillés sur plusieurs années. Ces vitesses sont assez précises pour prédire le trafic de certains jours types sur certains tronçons. Des modèles sont alors élaborés pour créer des fichiers journaliers de vitesses historisées : pour chaque jour, chaque tronçon de voirie possède autant de vitesse que de minute de la journée, organisée dans une structure de type *array*. Lors de l'évaluation d'un arc à partir d'un nœud et d'un horaire, il est possible d'utiliser la vitesse correspondant à cet horaire. S'il existe un conflit entre les vitesses temps réel, prédictive et historisée, la priorité est au temps réel, puis prédictif, puis historisé.

Vitesses statiques pénalisées *Graphe routier.* Vu que les vitesses des axes secondaires ne sont pas mesurées, si un axe est bloqué par un très fort trafic, le calculateur aura tendance a éviter ces tronçons au profits de chemins annexes. Or en réalité, il est courant que ces chemins soient également bloqués. Par conséquent, pour éviter cela, et aussi pour sur-estimer plutôt que sous-estimer les temps de trajet en voiture, l'algorithme pénalise par un coefficient multiplicatif tous les coûts des arcs qui ne sont pas fournis en vitesse temps réel, prédictive ou historisée.

Priorité par classes de route *Graphe routier.* Une classe de route est une notion définie par Navteq qui correspond à l'importance de la route sur le réseau. Une autoroute ou une voie rapide a une classe plus importante que les nationales et les départementales. Il est possible de donner priorité aux classes élevées afin d'éviter les raccourcis non désirés par des petites routes de lotissement, des ruelles étroites, etc. Cela est fait par le biais d'un coefficient multiplicatif lié à chaque classe, appliqué aux poids des arcs des tronçons correspondant.

Pénalisation de la pente *Graphe vélo.* Il est possible de disposer d'information d'altitude sur les tronçons de voirie. Dans ce cas, ils sont utilisés pour déterminer une pente du tronçon, positive ou négative. L'algorithme peut alors prendre en compte cette information pour favoriser des arcs à pente nulle ou négative.

Pistes cyclables *Graphe vélo.* Dans les données cartographiques, certains tronçons sont marqués comme des pistes ou bandes cyclables. Ces chemins peuvent être privilégiés par l'algorithme par le biais d'une pénalité multiplicative pour les tronçons non cyclables. Cette pénalité est plus ou moins grande en fonction du niveau de « sécurité », spécifié à la requête, que désire l'utilisateur.

Pénalisation des zigzags *Graphe piéton et vélo* Certains chemins plus courts font de nombreux zigzags, changeant de voirie souvent et pouvant causer chez l'utilisateur de la confusion. Lorsque deux trajets sont quasiment similaires en termes de durée et de temps, le calculateur favorise celui qui change de rue le moins souvent. Pour cela, il utilise lors de l'évaluation d'un arc une très faible pénalité additive si l'arc appartient à une rue différente de la précédente.

Route bloquée *Graphe routier.* Une fonctionnalité importante du calculateur est la possibilité de perturber certains tronçons de voirie à certains horaires. Ces perturbations bloquent toute navigation, et une évaluation d'arc perturbé à un horaire correspondant sera rejeté.

Déviations *Graphe routier.* Il est possible d'associer une déviation à toute perturbation routière, déviation que les automobilistes bloqués par la perturbation sont supposés emprunter en lieu et place des tronçons bloqués. Une déviation est définie par une série de tronçons très fortement favorisés par l'algorithme s'il se trouve que le trajet optimal aurait du passer par la route bloquée. Cela est fait en deux étapes : d'abord le trajet optimal est calculé sans prendre en compte les perturbations, puis s'il se trouve qu'un tronçon perturbé se trouve sur celui-ci, un calcul est relancé en bloquant les tronçons perturbés, et en marquant les tronçons de la déviation. Lorsque les tronçons marqués sont évalués, un coefficient inférieur à un est utilisé pour diminuer leur poids.

Trajet le plus accessible *Graphe piéton* Une dernière fonctionnalité consiste à ajouter un critère d'accessibilité lors des recherches d'itinéraire piéton. Certaines voiries ne sont pas accessibles facilement pour les fauteuils roulant, et certains clients fournissent une liste de voiries aménagées. Celles-ci peuvent être favorisées lors du calcul d'itinéraire si la requête l'impose.

2.2 Requêtes n-m et isochrones

Comme pour les calculs en transport en commun, il est possible de spécifier plusieurs lieux de départ, ou plusieurs lieux d'arrivée, à une requête dans un graphe individuel. Cela est utile pour les trajets combinés, comme décrit section 3. Il n'est pas question ici de trouver tous les trajets plus rapides commençant par chacun des points de départ, mais le meilleur trajet parmi toutes les combinaisons possibles d'arrêts de départ et d'arrivée spécifiés. Cela est fait de la même manière que sur le graphe transport en commun.

Requête 1-n, ou isochrone Il arrive qu'au lieu d'une recherche de trajet optimal entre deux points, il soit demandé l'ensemble des durées de tous les trajets optimaux du (ou des) point(s) de départ spécifiés vers une partie des lieux géographiques du graphe. L'algorithme utilisé est inchangé, seule la sortie est modifiée : une liste des lieux géographiques atteints avec la durée du trajet. Notons que ce type de requête a bien souvent un rayon de recherche maximum afin de limiter les temps de calcul sur les graphes de grande taille.

Nous remarquons donc que, bien que le calcul soit tout à fait élémentaire dans son principe de base, il existe de très nombreuses contraintes de modélisations et de requêtes qui le complexifient.

3 Rabattement individuel sur transport en commun

Nous allons évoquer ici la méthode employée par Cityway pour calculer des trajets multimodaux qui suivent le schéma suivant :

- transport individuel (marche, vélo ou voiture),
- transport en commun (incluant les correspondances marche à pied),
- transport individuel (marche, vélo ou voiture).

Nous cherchons à résoudre la requête $\mathsf{A}@\tau \rightarrow \mathsf{B}$ avec A et B des lieux géographiques, en autorisant les trois étapes décrites ci-dessus, avec un rayon maximal de transport individuel Δ.

L'algorithme que nous explicitons ici est un algorithme distribué, comme décrit dans la section 3.3.2 du chapitre I.

3.1 Principe de résolution

L'algorithme utilisé est composé des trois étapes suivantes :

1. **Calcul 1-n** sur le graphe individuel à partir du lieu A à l'horaire τ, en *départ à*, vers tous les lieux de type ARRÊT. Nous ne recherchons que des arrêts physiques à « proximité » de A : le calcul est borné par la distance maximale Δ. Une liste d'arrêts avec leur temps d'accès est obtenue ainsi.
2. **Calcul m-1** sur le graphe individuel à partir du lieu B, en *arrivée à*, sans horaire d'arrivée précisé, vers tous les lieux de type ARRÊT et borné par la distance maximale Δ. Les graphes individuels sont dans la majorité des cas indépendants de l'horaire, l'absence d'un horaire d'arrivée ne pose donc pas de problème. L'algorithme calcule ainsi une liste d'arrêts à proximité de B avec les temps d'accès de ces arrêts vers B.
3. **Calcul n-m** sur le graphe TC en *départ à*. Chaque arrêt atteint dans l'étape une donne une liste de nœuds de marche de départ, avec un horaire de départ spécifique, et un coût d'initialisation non nul correspondant au temps d'accès.

Chaque arrêt atteint dans la deuxième étape donne la liste des nœuds de marche d'arrivée, sans horaire d'arrivée mais avec un coût supplémentaire correspondant au temps d'accès. Ce coût supplémentaire est ajouté au coût des labels de fin arrivant à un nœud final. Le label de fin de moindre coût est comme d'habitude choisi pour déterminer le meilleur itinéraire TC.

Une dernière étape consiste à reconstituer les trois étapes de l'itinéraire optimal, à partir du meilleur trajet trouvé dans la troisième étape.

Illustration de l'algorithme distribué La figure 18 illustre cet algorithme. Nous la détaillons ci-après.

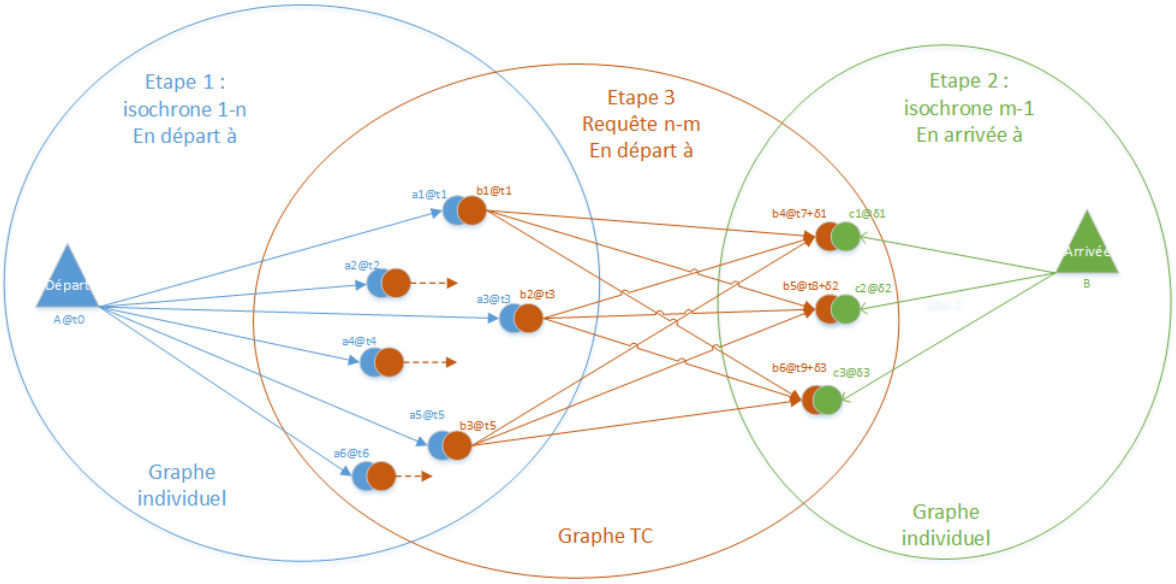


FIGURE 18 – Calcul combiné marche/TC/marche

1. Les nœuds du graphe individuel $a_1, \ldots, a_6$, sont atteints aux horaires $t_1, \ldots, t_6$ à partir de l'évènement de départ $A@\tau$.
2. Les nœuds c_1, c_2, c_3 permettent d'atteindre B, avec des durées de $\delta_1, \delta_2, \delta_3$.
3. Les nœuds de marche $b_1, \ldots, b_6$ correspondent aux arrêts des nœuds $a_1, \ldots, a_6$ et sont utilisés comme nœuds de départ du calcul n-m sur graphe TC. Les nœuds de marche b_4, b_5, b_6 correspondent au nœuds c_1, c_2, c_3 et sont utilisés comme nœuds de fin du calcul n-m. Les horaires d'arrivée aux nœuds de fin, ajoutés aux temps d'accès trouvé dans la deuxième étape sont $t_7 + \delta_1, t_8 + \delta_2, t_9 + \delta_3$. L'un des trois itinéraires est sélectionné comme étant le meilleur.
4. L'itinéraire global combiné est reconstitué à partir de l'itinéraire TC sélectionné.

Étape graphe *reverse* En pratique, une étape supplémentaire est effectué entre la troisième et quatrième étape sur le graphe TC graphe *reverse*. En effet, il est important d'optimiser le critère **lexicographique** plutôt que le simple critère d'**arrivée au plus tôt**. Pour cela, les arrêts d'arrivée, leurs horaires trouvés dans la troisième étape et

les temps d'accès trouvés dans la deuxième étape sont utilisés comme points de départ du calcul graphe *reverse*. Une fois le meilleur trajet trouvé, il n'est pas nécessaire de recalculer les trajets de rabattement en transport individuel vu qu'il ne dépendent pas du temps.

Départ au plus tard Le problème $\mathsf{A} \rightarrow \mathsf{B}@\tau$ est symétrique et se résout avec un algorithme similaire, que nous ne détaillerons pas.

3.2 Spécificités du calcul combiné

Faire un calcul combiné entraine des spécificités sur l'algorithme combiné lui-même, ou sur les algorithmes exécutés sur les graphes TC ou individuels, que nous allons évoquer ici. Ces difficultés se rajoutent à toute celles mentionnées précédemment, et contribuent à la complexité du calcul multimodal.

Milliers d'arrêts Si le rayon de transport individuel est élevé dans une zone de transport en commun dense, il peut y avoir des milliers d'arrêts trouvés lors des étapes de calcul 1-n. Le calculateur doit retenir les milliers de chemins détaillés en mémoire, afin de pouvoir reconstituer la réponse finale après l'étape de calcul n-m en transport en commun.

Séparation des arrêts Il peut arriver que certains arrêts soient présents à la fois en rabattement à partir de A et vers B. Lorsque cela arrive, ces arrêts sont triés et désignés ou bien comme arrêts de départ, ou bien comme arrêts d'arrivée, mais pas les deux, en fonction de leur distance à A et B.

Pénalité sur les rabattements Nous avons vu que les temps d'accès des étapes de rabattement donnent des coûts supplémentaires aux arrêts de départ et d'arrivée du calcul TC. Ces coûts sont augmentés par un coefficient multiplicateur, en général égal à C_w (coefficient utilisé pour pénaliser les correspondances marche à pied). Cela a pour effet de privilégier des trajets avec des rabattements individuels courts et des plus grandes étapes de transport en commun. Si le coefficient est fixé à 1 (pas de pénalité), il est très fréquent que les rabattements aient une distance parcourue proche du maximum possible, c'est-à-dire du rayon de recherche spécifié.

Changements sur le graphe TC Nous avons vu dans la section 1 que l'algorithme sur le graphe TC ignore les temps de transfert minimum pour les arrêts de départ et d'arrivée du trajet TC. Cependant lorsqu'ils sont précédés ou suivis de rabattements à partir de lieu de départ et d'arrivée, ces temps de transfert sont appliqués normalement. En effet, si l'on arrive à pied, en vélo ou à voiture à un arrêt, il est plus prudent d'être sûr que l'utilisateur a une marge confortable d'attente avant de prendre le transport en commun.

Deuxième étape dépendante à l'horaire Dans les cas rares où la deuxième étape se fait sur le *graphe routier* avec des informations de trafic en temps réel ou historisé, celles-ci ne sont pas utilisées. En effet, le calcul m-1 est effectuée sans

connaître l'horaire d'arrivée, il n'est donc pas possible de connaître les vitesses moyennes, qui dépendent de l'horaire au nœud de l'arc évalué.

4 Discussion

Cette dernière section est consacrée à l'analyse comparée de l'existant de la littérature et de l'existant Cityway, afin de mettre en exergue les choix d'orientations des travaux de cette thèse.

Nous avons vu d'une part que l'état de l'art sur les questions de plus court chemin dans des réseaux de transport individuels ou en commun est très complet. Une grande partie des recherches est issue d'évolutions itératives de l'algorithme de Dijkstra, en particulier pour accélérer toujours plus efficacement les temps de calcul. Ces nombreuses techniques d'accélération nécessitent parfois des temps de prétraitement assez lourds. Une autre partie de la recherche est consacrée aux questions multicritères et aux questions multimodales.

D'autre part, le calculateur de Cityway est un logiciel dont les algorithmes sont des variantes de l'algorithme de Dijkstra et du calcul distribué sur plusieurs graphes. Par conséquent, il est élémentaire d'un point de vue algorithmique, mais il n'en demeure pas moins extrêmement riche et complexe du point de vue des paramètres, des requêtes, des contraintes et des critères pris en compte. La proximité des modèles de graphe utilisés avec la réalité donne une très bonne flexibilité au calculateur : il est aisé d'intégrer de nouvelles contraintes ou de nouveaux critères à la demande des clients. Nous aimerions conserver cette flexibilité. Néanmoins, l'algorithme actuel souffre de plusieurs défauts, comme par exemple celui d'utiliser une somme pondérée comme critère d'optimisation. Aussi, ne conserver qu'un label par nœud ne permet pas de renvoyer plusieurs trajets équivalents au trajet optimal trouvé.

4.1 Problèmes liés à l'algorithme actuel

Somme pondérée Un des problèmes que nous rencontrons avec le calculateur est que les divers critères sont agrégés comme une **somme pondérée**. Chaque critère, en dehors de la durée du trajet, influe sur le coût des labels par des poids supplémentaires ou des coefficients multiplicateurs. Cela implique que le plus court chemin au sens de ce coût n'est pas nécessairement le chemin le plus rapide au sens de la durée. Or, l'algorithme de Dijkstra ne fixe qu'un label par nœud, pas nécessairement le trajet d'horaire d'arrivée minimal. De plus, un horaire minimal permettrait à un label de s'étendre plus facilement, puisqu'arriver plus tôt à un arrêt permet en général d'emprunter plus de courses. Une course figurant dans le trajet optimal recherché peut être omise par l'algorithme pour cette raison.

En ce sens, un exemple élémentaire est l'illustration suivante. Supposons que nous cherchions à résoudre le problème $\mathsf{A}@\tau \rightarrow \mathsf{B}$ avec $\tau = 8\text{h}00$. Supposons aussi que B ne soit relié au réseau de transport en commun que par une ligne de car partant de C. Supposons également que cette ligne n'ait qu'une seule course journalière, partant à

8h30. Alors, tout trajet de A vers B doit nécessairement passer par C, est n'est réalisable que si son horaire à C est antérieur à 8h30. Imaginons maintenant que l'algorithme de Dijkstra avec somme pondéré fixe le label sur C avec un trajet direct partant de A et arrivant à 8h32. La correspondance est impossible et l'algorithme se termine sans avoir trouvé de solution. Or il pourrait exister un trajet de A vers C comprenant un changement, ou un peu plus de marche à pied, dont le coût est plus élevé que le trajet direct, mais arrivant plus tôt et permettant d'effectuer la correspondance.

Cet exemple est illustré par la figure 19, avec des pénalités additives de montée et descente de cinq (dans cet exemple, nous considérons que les temps de transfert minimums sont nuls). Les étiquettes indiquent les coûts des meilleurs labels trouvés à la fin de l'algorithme, avec le label de coût 45 qui n'est pas assez bon pour remplacer le label d'horaire antérieur mais de coût 42.

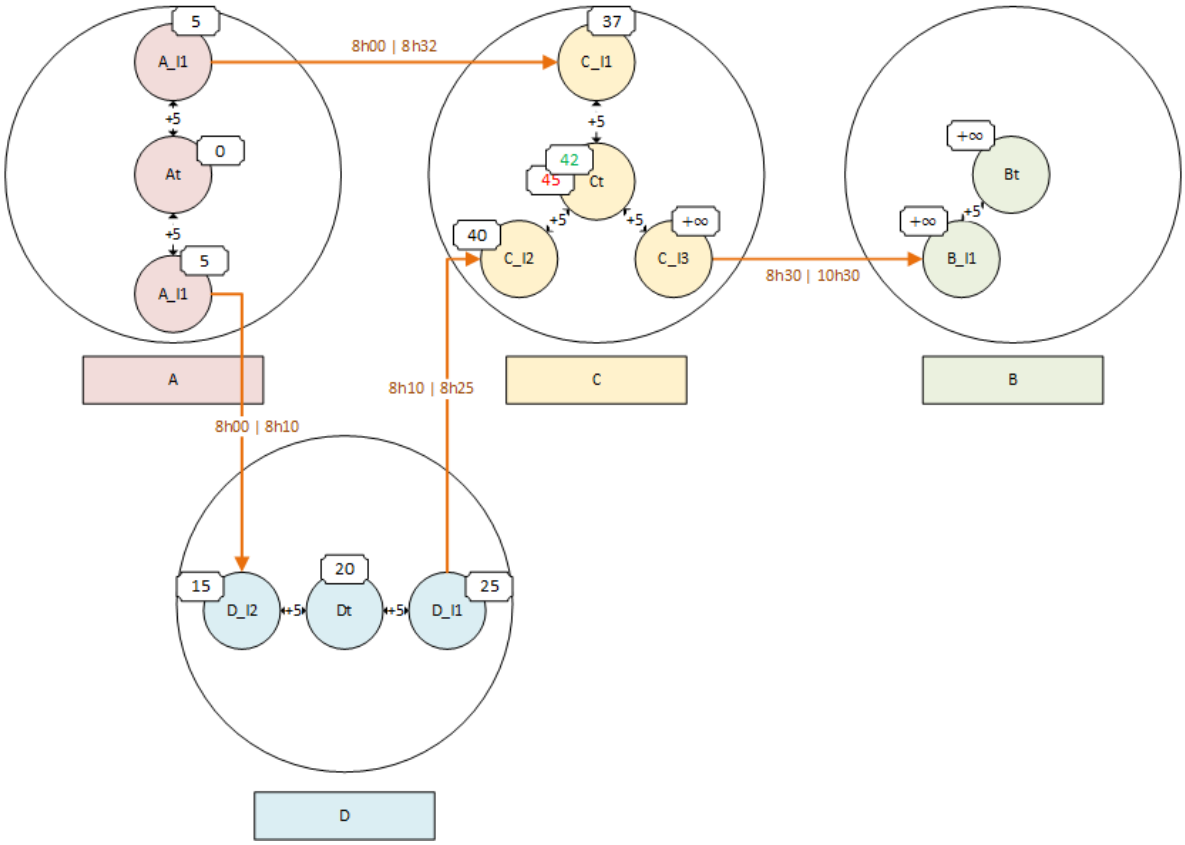


FIGURE 19 – Trajet réalisable omis à cause de la somme pondérée

Correspondances multiples en monolabel Un autre problème est dû au fait qu'il n'y a qu'un label par nœud et qu'il est interdit de renvoyer des trajets avec plusieurs correspondances en marche à pied successives. En effet, il est possible qu'un label arrivant du nœud ITI A vers un nœud MARCHE M_0 soit rejeté au profit d'un meilleur label arrivant d'un autre nœud MARCHE M_1. Dans ce cas, toute correspondance marche à pied à partir de M_0 sera interdite. C'est un problème si le trajet optimal utilise l'arc (A, M_0) et continue par un arc CORRESPWALK (M_0, M_2), il ne pourra pas être obtenu par notre algorithme. La figure 20 illustre ce cas sur un réseau réel.

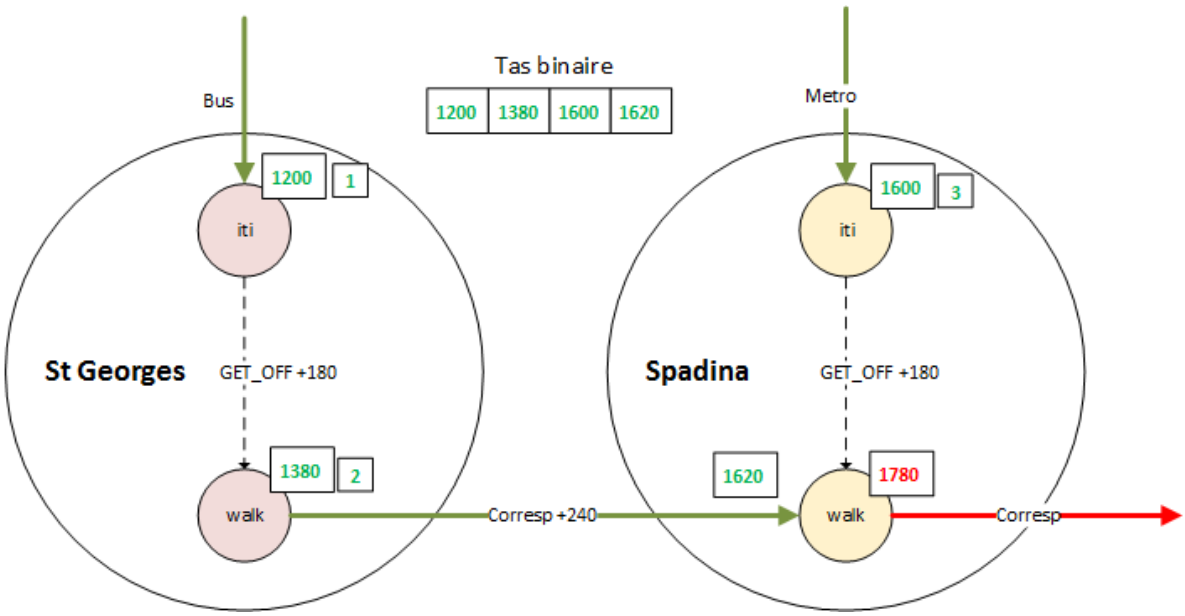


FIGURE 20 – Interdictions des correspondance multiples

4.2 Axe de recherche choisie pour cette thèse

Même si les nombreuses techniques d'accélération semblent attrayantes, compte tenu de leurs performances exceptionnelles comparativement à l'algorithme utilisé par le calculateur de Cityway, leur utilisation implique bien souvent des processus de prétraitement assez lourds, qui diminuent grandement la flexibilité du calculateur. Comme les calculateurs déployés peuvent concerner de très nombreux transporteurs, et comme chaque transporteur met très fréquemment ses données à jour, cela entraine la génération de nouveaux graphes et de nouveaux fichiers horaires régulièrement. Par conséquent, les temps de prétraitement ne doivent pas être trop longs. Enfin, les algorithmes de prétraitement fonctionnent mal ou pas du tout avec les données temps réel et prédictives dont nous disposons.

De plus, certains algorithmes sans prétraitement fonctionnent assez mal avec les réseaux de transport en commun, en particulier les algorithmes « goal-oriented » type A*.

Vu que les temps de calcul de l'algorithme actuel ne dépasse pas la demi-seconde pour les réseaux traités les plus grands, nous avons décidé de ne pas nous intéresser à leur accélération dans un premier temps.

Conserver le modèle *time-dependant*, proche de la réalité, avec un algorithme simple comme celui de Dijkstra permet de traiter les diverses spécificités que nous avons décrites dans ce chapitre, et d'autres que l'on pourrait nous demander, sans grande difficulté. Le code du calculateur est complexe et riche mais assez fragile : changer la modélisation des données ne nous parait pas judicieux, dans un souci de stabilité et de maintenabilité.

Ainsi, nous avons fait le choix de concentrer nos efforts sur l'**aspect multicritère du problème**, et de faire évoluer le calculateur d'itinéraire vers une version **multilabel**

souple et efficace. En effet, la qualité des trajets obtenus, au sens de la satisfaction des utilisateurs, est un point crucial dans l'appréciation du produit par les clients de Cityway. De plus, la maitrise d'une telle technique permet potentiellement de traiter les divers problèmes auxquels la version monolabel ne peut faire face. La souplesse du code multilabel permet de l'adapter à de nombreux critères.

Retenons néanmoins que le recul que nous avons pu prendre en étudiant l'état de l'art restera très utile dans des perspectives à plus long terme d'optimisation des temps de calcul ou d'adaptation d'algorithmes et modélisations existants.

Chapitre IV

Algorithme multilabel pour une optimisation multicritère et application à un cas bicritère

Sommaire

Ce chapitre est consacré à l'explication détaillée de nos travaux sur le calculateur pour passer d'un calcul monolabel basé sur un algorithme de Dijkstra à un calcul multicritère dont les résultats constituent un front de Pareto. Pour cela, nous expliquons ce qu'est l'optimisation multicritère, et comment elle s'applique au calcul d'itinéraire, avec une brève revue de littérature en section 1. Puis nous détaillons les algorithmes utilisés par le calculateur de Cityway avant (section 2) et après (section 3) modifications. Enfin, nous appliquons l'algorithme au cas de deux critères , le plus rapide et le moins de changements, en détaillant les expérimentations menées pour tester cette évolution (section 4).

1 État de l'art

Comme introduit dans le chapitre I, une recherche d'itinéraire dans un réseau de transport en commun cherche à optimiser plusieurs critères : l'horaire d'arrivée, le nombre de changements et la durée de marche à pied dans le trajet par exemple. Nous introduisons dans cette section les divers concepts liés à l'optimisation multicritère dans la littérature. Les définitions et les exemples se réfèrent au livre [Ehrg2006], qui traite du sujet.

L'optimisation multicritère ou multiobjectif consiste à choisir les solutions d'un problème en se basant sur plusieurs attributs, indépendants ou contradictoires, que le décideur aura jugé assez important pour discriminer les solutions.

1.1 Définitions

Définissons formellement les les notions relatives à l'optimisation multicritère dont nous avons besoin dans ce chapitre.

Définition 30 *On appelle* ***critère*** *une fonction de l'espace des solutions réalisables vers n'importe quel ensemble totalement ordonné, le plus souvent* $\mathbb{R}$.

Ainsi, avec S_1 et S_2 deux solutions de $\mathcal{S}$, un critère $f : \mathcal{S} \to \mathbb{R}$ permet de préférer S_1 à S_2 ssi $f(S_1)\ R\ f(S_2)$, ou R est $<$ ou $>$ selon que l'on veut minimiser ou maximiser ce critère. Nous considérons pour la suite de ce document que tous les critères doivent être minimisés et donc que R est $<$.

Définition 31 *Pour un ensemble fini de critères* F*, on dit que* S_1 ***domine*** S_2 *si* $\forall f \in F$, $f(S_1) \leq f(S_2)$ *et* $\exists f \in F \,|\, f(S_1) < f(S_2)$. *On écrira* $S_1 \prec S_2$

Définition 32 *On dit que* S *est une solution* ***Pareto-optimale*** *ou* ***efficace*** *si pour toute autre solution réalisable* S', $\neg(S' \prec S)$, *c'est-à-dire qu'il n'existe aucune solution réalisable qui domine* S.

Définition 33 *Soit un problème d'optimisation multicritère à* k *fonctions critères* $f_1, \ldots, f_k$ *et* $\mathcal{S}_P$ *l'ensemble de ses solutions efficaces. Le* ***front de Pareto*** *du problème est l'image de* $\mathcal{S}_P$ *par* $(f_1, \ldots, f_k)$, *inclus dans* $\mathbb{R}^k$.

Par abus de langage, nous appelons dans ce chapitre front de Pareto indifféremment l'ensemble des solutions efficaces $\mathcal{S}_P$ et leur image par les fonctions critères.

1.2 Exemple

Extraite de [Ehrg2006], la figure 21 montre les quatre solutions réalisables au problème du choix d'une nouvelle voiture, en prenant en compte les critères du prix et de la consommation d'essence. Ces critères doivent être minimisés. La solution Toyota est dominée par Ford et Opel, qui consomment moins et coûtent moins cher, VW est dominée par Opel, mais pas par Ford. Ford et Opel ne sont dominées par aucune solution, et forment donc le front de Pareto du problème.

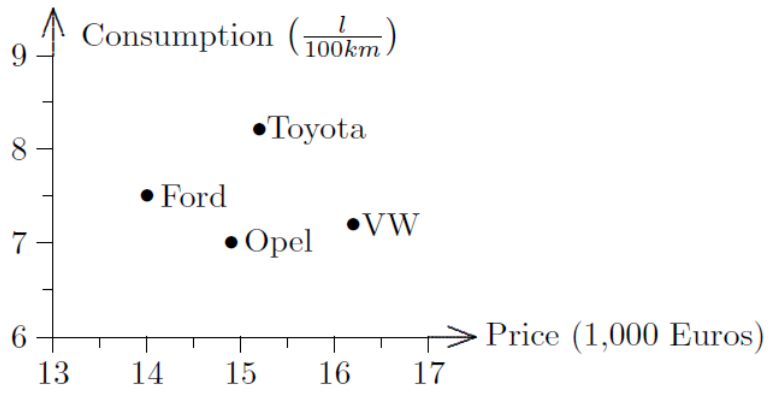


FIGURE 21 – Exemple : solutions réalisables d'un problème à deux critères

1.3 Optimalité

Selon le problème à résoudre, et les critères choisis, un front de Pareto peut compter un nombre plus ou moins grand de solutions, dont certaines seraient plus ou moins pertinentes. Par exemple, une solution légèrement meilleure sur un critère important et vraiment moins bonne sur un critère secondaire sera très souvent laissée de côté en pratique. Trouver des solutions efficaces selon tous les critères mentionnés par le décideur ne suffit donc pas forcément à résoudre le problème.

Lorsque l'on dispose des informations supplémentaires sur l'importance relative des critères du problème d'optimisation, il devient possible de redéfinir le problème traité et de sélectionner parmi les solutions du front de Pareto celles qui nous intéressent. De plus, le problème peut parfois être ramené à un problème monocritère, et donc plus rapide à résoudre en temps d'exécution. Ainsi, il n'est pas nécessaire de rechercher l'intégralité du front de Pareto.

Nous présentons ici quelques méthodes globales pour orienter la résolution du problème.

Tous les chemins optimaux La première méthode consiste à trouver tous les chemins Pareto-optimaux, et les filtrer *a posteriori*.

Ordre lexicographique Si les critères peuvent être ordonnés par importance, on peut utiliser cet ordre lexicographique pour choisir la solution optimale, qui sera alors unique.

Somme pondérée Si les critères peuvent être comparés entre eux, il est possible d'utiliser une somme pondérée de critères $\sum_{i=1}^{k} \lambda_i f_i(S)$, que l'on cherche à minimiser.

1.4 Algorithme de Martins

Nous avons fait le choix de mentionner l'algorithme de Martins [Mart1984] ici car c'est celui que nous avons choisi pour développer la version multilabel du calculateur.

C'est un choix évident : le calculateur est basé sur l'algorithme de Dijkstra, et l'algorithme de Martins est une extension naturelle de celui-ci permettant de trouver les chemins efficaces dans un cadre multicritère.

C'est un algorithme écrit par Ernesto Queirós Vieira Martins, dont l'objectif est de trouver toutes les solutions d'un problème d'optimisation de plus court chemin multicritère dans un graphe. Il est de complexité exponentielle en général, étant donné que le nombre de solutions du front de Pareto peut être exponentiel en fonction du nombre de nœuds, et ce même pour deux critères.

L'idée de l'algorithme est de ne plus considérer des nœuds mais des étiquettes associées à chaque nœud, qui ont pour information l'étiquette précédente et les valeurs des critères sur le nœud pour ce label. Il existe plusieurs variantes de l'algorithme : il peut boucler soit sur le nœuds et étendre toutes les étiquettes non dominées, soit directement sur les étiquettes, contenues dans une liste ou dans un tas binaire ordonné par une relation d'ordre total. L'algorithme s'arrête lorsque toutes les étiquettes créées ont été étendues. A chaque étape, les étiquettes dominées par des étiquettes nouvellement étendues sont supprimées.

C'est l'algorithme utilisé dans la version multicritère du calculateur, décrit dans la section 3.

1.5 Revue de la littérature sur le calcul d'itinéraire multicritère

Un certain nombre d'articles et de travaux de thèse traitent de l'optimisation multicritère d'un itinéraire dans un réseau de transport en commun ou dans un réseau de voirie. Nous exposons ici les contributions les plus récentes et significatives, en lien avec ce travail de thèse, sur ces questions.

1.5.1 Comparaison d'algorithmes multicritère sur graphes *time-dependant* et *time-expanded*

Dans ses travaux, Schulz ([Schu2005]) fait une série d'expérimentations pour comparer l'efficacité de divers algorithmes sur les graphes *time-expanded* et *time-dependant* du réseau de voie ferré longue distance de l'Allemagne. Quels que soient les critères considérés (monocritère, bicritère ou lexicographique), le graphe *time-dependant* est plus intéressant que le graphe *time-expanded* du point de vue des temps de calcul, de facteurs variant de 1,5 à 3,3.

1.5.2 Dominance avancée

Les travaux de Schnee dans [Schn2009] concernent un calculateur d'itinéraire multimodal appelé MOTIS, développé sur un graphe *time-expanded*. En plus d'optimiser les calculs de front de Pareto pour avoir des temps de réponse acceptables d'un point de vue industriel, et de se pencher brièvement sur les calculs de trajets les moins chers,

robustes ou alternatifs, les travaux définissent les concepts intéressants de dominance relâchée et dominance restreinte. Ce calculateur est basé sur une version *time-expanded* du graphe, mais des expérimentations sont faites sur une version *time-dependant*, et l'auteur conclut que les temps de calculs sont similaires, à cause du temps mis à évaluer les coûts des arcs à la volée, mais que la consommation en mémoire est beaucoup plus grande dans le cas *time-expanded*, comme attendu.

La **dominance relâchée** consiste à n'appliquer une dominance d'une solution S_1 sur S_2 uniquement si elle la domine au sens classique, et si elle respecte d'autres contraintes, comme une marge m positive dans l'inégalité sur les critères : $f_k(S_1) + m_i(S_1, S_2) \leq f_k(S_2)$. Autre exemple, une solution train de nuit n'est pas comparable à une solution de jour, donc le label sera conservé même si la solution de jour est plus rapide ($m = \infty$ si on compare deux solutions incomparables). C'est un concept qui permet de garder des labels intéressants d'un point de vue de l'utilisateur qui auraient autrement été laissés de côté. Cela permet aussi d'attendre un point ultérieur du calcul pour discriminer deux labels.

La **dominance restreinte** est l'inverse de la dominance relâchée, dans le sens où la relation de dominance est moins restrictive. Cela permet par exemple de supprimer des labels plus courts de quelques minutes qu'un autre, mais avec de nombreux changements supplémentaires, solutions souvent inintéressantes. Attention toutefois, la relation peut ne plus être transitive, en fonction du choix de $m_i(S_1, S_2)$, cette fois négatif. Il faut que la relation de dominance reste transitive pour que l'algorithme de Martins reste exact.

1.5.3 Multicritère sur graphe simplifié

Geisberger expose dans [Geis2010] un modèle de graphe simplifié ne comportant qu'un nœud par arrêt physique et aucun arc parallèle. Il définit alors une relation de dominance entre les labels de manière à prendre en compte les temps de changements minimum à chaque arrêt et à ne pas dominer les labels dont le prolongement ne peut pas être remplacé. Un label en domine un autre à un arrêt s'il est meilleur en termes de temps, mais aussi si tout changement permis par le deuxième est également permis par le premier. Cela se traduit par des comparaisons sur les temps d'attente des trains à l'arrêt et le temps minimum nécessaire à un changement à cet arrêt, ainsi que le temps nécessaire pour aller d'un label à l'autre. Le lecteur intéressé pourra approfondir le sujet dans le chapitre 4 de la thèse de Geisberger. L'intérêt de cette modélisation est que malgré l'augmentation du nombre de labels par arrêt, le graphe est beaucoup plus petit et le temps de calcul par rapport à l'application de l'algorithme de Dijkstra dans un graphe *time-dependant* classique est réduit d'un facteur cinq.

1.5.4 Multicritère et *Transfer Patterns*

Pour son travail de précalcul sur les *Transfer Patterns*, vu précédemment dans le chapitre II section 4.5, Geisberger [Geis2010] utilise le calcul d'itinéraire multicritère sur graphe *time-expanded*. Les critères choisis sont d'une part le temps et d'autre part la somme des pénalités de montée et descente aux arrêts sur lesquels sont effectués des changements. Il cherche ainsi à minimiser le temps perdu dans les changements plutôt

que leur nombre. Il explique cependant que son implémentation est assez souple pour pouvoir changer de critère.

1.5.5 Multicritère et RAPTOR

Dans [Dell2012], Delling étend son algorithme de programmation dynamique RAPTOR présenté dans le chapitre II, section 4.6 au cas multicritère : McRAPTOR, qu'il teste avec le critère du moins cher. A la place de stocker et comparer des temps τ_k à chaque arrêt et pour chaque itération (k *round*), l'algorithme compare une liste de labels B_r nouvellement calculés liés à l'itinéraire en cours de lecture à des listes de labels fixées auparavant B_{k-1} et les fusionnent dans une liste de labels non dominés B_k. Les résultats obtenus sur un calcul à trois critères (heure d'arrivée, prix et nombre de changements) sont très bons, puisque quatre fois plus rapides que l'algorithme de Martins.

Cet algorithme est étendu au cas multimodal dans [Dell2013], avec un mélange de McRAPTOR pour le transport en commun et de Martins pour la partie marche à pied entre les arrêts sur un graphe voirie.

1.5.6 Utilisation de bornes

Parmi les méthodes utilisées pour optimiser le calcul de plus court chemin bicritère, Tung introduit dans [Tung1992] la possibilité de calculer les bornes inférieures sur un critère et supérieures sur l'autre pour aller d'un nœud à la destination, avec un calcul monocritère partant de l'arrivée, optimisant le premier et stockant les valeurs du second. Deux calculs monocritères nous donnent donc toutes les bornes nécessaires. Puis il est possible de couper des labels dont on sait qu'ils ne seront pas intéressants grâce à ces bornes, lors du calcul multicritère.

Le même principe est appliqué dans [Sauv2011] pour un calcul d'itinéraire en vélo. Le problème de cette méthode dans un réseau *time-dependant* est que le calcul en sens inverse ne peut être fait sans connaissance préalable de la date d'arrivée.

2 Algorithme monolabel existant

Nous détaillons dans cette section l'algorithme monolabel de la version actuelle du moteur de l'entreprise Cityway dans le réseau de transport en commun. Nous allons omettre les calculs liés aux graphes individuels, et décrire l'algorithme répondant à la requête d'**arrêt à arrêt** en *départ à* $A@\tau \rightarrow B$, sans rabattement en marche à pied.

Le calculateur effectue un algorithme de Dijkstra sur le graphe transport en commun en utilisant un tas binaire trié par coût agrégé, en respectant les règles d'extension des labels, de dominance entre labels, de labels de départ, et de condition de fin de calcul qui feront chacun l'objet d'une section.

L'algorithme utilise les paramètres de configuration P_t, P_c et C_w introduits dans le chapitre III. Pour des temps de changement réalistes, l'algorithme est paramétré par des

temps de montée et descente minimum P_t, dépendant du mode ou de l'arrêt sur lequel ils s'appliquent. Afin de minimiser le nombre de changements et la durée de marche à pied, les deux paramètres P_c et C_w sont utilisés. Rappelons que P_c permet de pénaliser chaque changement d'un coût additif sur les arcs GETIN et GETOFF, dépendant du mode du nœud ITI. Rappelons également que C_w est un coefficient multiplicatif qui s'applique aux coûts des arcs CORRESPWALK. Si P_c est différent de 0, ou si C_w est différent de 1, le coût d'un label sera différent de sa durée $(t - \tau)$, et l'algorithme sera sous-optimal, comme nous l'avons vu dans la section 4 du chapitre III.

La requête monolabel peut être modulée avec un critère d'optimisation qui prendra l'une des trois valeurs suivantes : plus rapide, moins de changements, moins de marche à pied. Le critère choisi impactera sur les valeurs des coefficients de coût P_c, C_w.

Notons que la résolution d'une requête n-m **d'arrêts à arrêts** en *départ à*, s'écrivant $((A_1@\tau + \delta_1), ..., (A_n@\tau + \delta_n)) \rightarrow (B_1, ..., B_m)$, est brièvement détaillée dans le chapitre III. Elle est très similaire à l'algorithme que nous décrivons ici.

2.1 Tas binaire

Étant donné que l'algorithme utilise un **tas binaire minimum** pour traiter les labels dans l'ordre croissant de leurs coûts agrégés, il nous a paru nécessaire de décrire la structure d'un tas binaire, et les opérations qu'il permet d'effectuer, avec leur complexité.

Définition

Un tas binaire minimum est un arbre binaire quasi complet, et qui respecte la règle suivante : la valeur, aussi appelée étiquette ou clé, de chaque nœud est inférieure ou égale à celle de ces fils.

Exemple

Un exemple de tas binaire minimum est donnée en figure 22.

Opérations sur un arbre binaire

L'avantage d'utiliser une telle structure est de pouvoir trouver l'élément de valeur minimale en $\mathcal{O}(1)$, tout en ayant une insertion et une suppression d'élément en $\mathcal{O}(\log(n))$. En pratique, le tas binaire est implémenté comme un tableau d'éléments ; la racine de l'arbre est en index 1 ; pour chaque étiquette E d'index i, ses fils sont d'index $2i$ et $2i + 1$. Les opérations possibles sont les suivantes, en supposant que chaque élément elem du tas binaire référence une donnée, une valeur et son index dans le tas.

- **Rise**(elem) : répare la structure de tas en échangeant récursivement elem avec son parent si sa valeur est strictement inférieure à celle du parent. Également appelé

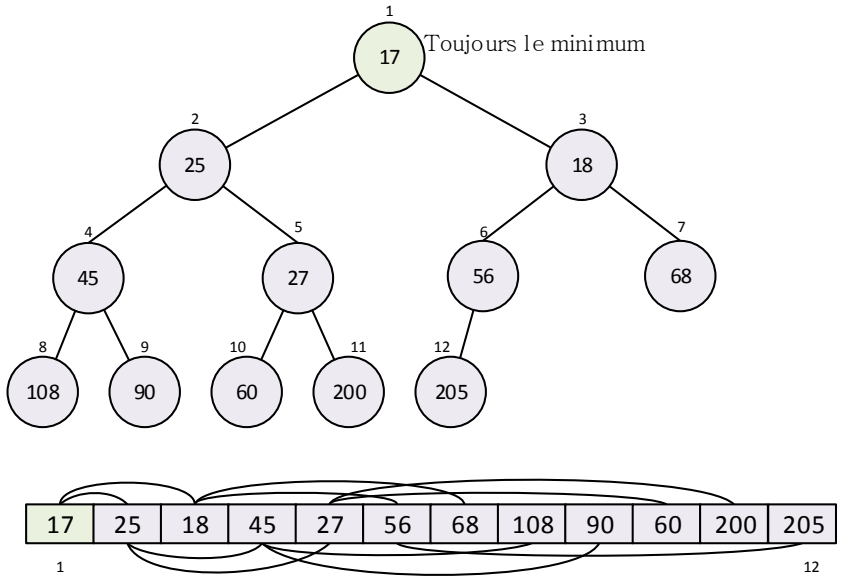


FIGURE 22 – Tas binaire - max

percolate up ou **bubble up**.

- **Sink**(elem) : répare la structure de tas en échangeant récursivement elem avec son fils d'étiquette maximale si sa valeur est strictement supérieure à celle-ci. Également appelé **percolate down**.
- **Insert**(data, val) : crée un elem de donnée data et de valeur val, l'insère comme dernière feuille puis appelle Rise(elem).
- **Delete**(index) : supprime un élément quelconque en l'échangeant avec le dernier élément du tableau puis en appliquant Rise ou Sink pour réparer le tas.
- **Decrease**(index, valeur) : réduit la valeur d'un élément puis applique Rise pour réparer le tas, le cas échéant.
- **FindMin**() : renvoie le premier élément du tableau.
- **IsEmpty**() : renvoie vrai si le tas est vide, faux sinon.

2.2 Règles du calcul monolabel

Le calcul monolabel suit un algorithme de Dijkstra, il n'y a donc qu'un label par nœud et le tas binaire contient des références directes au nœud. L'algorithme est décrit

en pseudo code (Algorithm 2) et détaillé ci-dessous.

En premier lieu, le label de chaque nœud est initialisé avec un coût égal à MAXEVAL (valeur maximale d'un int), et un temps non évalué (-1). Puis, le nœud de départ est mis à jour avec un coût nul et un temps τ. Une référence à ce nœud est également insérée dans le tas binaire avec comme valeur son coût nul. Le nœud d'arrivée est marqué comme nœud d'arrivée.

Une variable bestCost est introduite pour représenter le coût minimum d'un chemin menant du départ à l'arrivée. Cette variable est initialisée à la valeur MAXEVAL (entier maximum).

Le cœur de l'algorithme est constitué d'une boucle principale, qui ne s'arrête que lorsque le tas est vide, et dont la première instruction est de sortir l'élément minimum du tas pour le traiter. Une boucle imbriquée parcourt l'ensemble des arcs sortant du nœud traité et évalue le coût de chaque arc selon des règles d'extension de labels décrites ultérieurement. Si le coût calculé dépasse bestCost ou un coût déjà calculé sur le nœud nouvellement atteint, il est inutile d'étendre le label. Dans le cas contraire, ce nœud est mis à jour avec un nouveau coût. De plus, le tas binaire est mis à jour : soit par l'insertion d'un nouvel élément lié à ce nœud, de valeur le nouveau coût calculé, soit par la diminution de la valeur de l'élément existant lié à ce nœud.

Suite à cette boucle imbriquée, bestCost est mis à jour si le nœud traité est marqué comme nœud d'arrivée et si le coût fixé est inférieur à bestCost. Le label de ce nœud est gardé en mémoire comme étant le meilleur label final.

Une fois le tas binaire vide, le meilleur label final est renvoyé.

L'arrêt de l'algorithme est assuré car chaque nœud n'est traité qu'une fois au maximum : en effet, étant donné que le coût du nœud traité à chaque étape est croissant (propriété du tas binaire minimum) et que toutes les évaluations d'arc sont positives, on ne pourra pas retrouver un meilleur coût que celui déjà fixé avec une évaluation ultérieure.

Dans un souci de simplification, un certain nombre de traitements spécifiques à des cas particuliers ont été omis de cette description, comme par l'exemple la dominance du label avec le plus d'attente possible à coût égal (plus de temps pour permettre un changement), la duplication des labels sur les nœuds de fin de calcul qui sont évalués en suivant des règles spéciales (pas de pénalité de montée/descente, correspondance interdite si arrêt en rabattement de marche à pied...), ou encore une limitation possible du calcul en temps maximum autorisé et dont le dépassement arrête le calcul.

Ressources

L'algorithme monolabel cherche à optimiser une seule ressource :

- un coût $\mathtt{c}$, égal au temps total depuis l'horaire de départ, en plus de pénalités variées.

De plus, des attributs supplémentaires sont utilisés pour le calcul et la propagation des labels :

Algorithm 2 Algorithme monolabel

```
result ← vide
for all n ∈ nodes do
  n.Label.c ← MAXEVAL
  n.Label.t ← -1
end for
A.Label.Init()
Heap.Insert(A, 0).
B.IsFinishNode ← true
bestCost ← MAXEVAL
while !Heap.IsEmpty() do
  node ← Heap.DelMin()
  for all outlink ∈ node.Outlinks do
    e ← outlink.EndNode
    endLabel ← ExtendLabel(node, outlink)
    if endLabel.c < e.label.c and endLabel.c < bestCost then
      if e.label.c = MAXEVAL then
        Heap.Insert(e, endLabel.c)
      else
        Heap.Decrease(e, endLabel.c)
      end if
      e.label ← endLabel
    end if
  end for
  if node.IsFinishNode and node.label.c < bestCost then
    bestCost ← node.label.c
    result = node.label
  end if
end while
return result
```

- un temps t, correspondant à l'horaire actuel,
- la dernière course C empruntée par ce label.

Règle d'extension

La règle d'extension d'un label sur un arc est très dépendante de son type : arc GETIN, arc GETOFF, arc ITI ou arc CORRESPWALK. Rappelons que le chapitre III détaille le modèle de graphe utilisé et les différents types d'arcs et de nœuds.

Nous utilisons les notations suivantes lors de l'extension par un arc a d'un label L vers un label L' sur un nœud e.

- (c, t, C) les ressources du label L.
- (c', t', C') les ressources du label L'.
- M le mode du nœud e
- P_t pénalité en temps du mode M, éventuellement modifié par l'arrêt lié à e. En début ou fin de calcul, $P_t = 0$.
- P_c pénalité en coût du mode M, éventuellement modifiée par l'arrêt ou par la ligne liés à e.
- si a est de type CORRESPWALK, d est la durée de correspondance marche à pied.
- Les fonctions $\mathit{prochainHoraire}(t, e)$ et $\mathit{prochaineCourse}(t, e)$ renvoie le prochain horaire et la prochaine course sur le nœud e en cherchant à partir du temps t.
- La fonction $\mathit{horaire}(C, e)$ renvoie l'horaire de la course C au nœud e.
- Le scalaire C_w est un coefficient multiplicatif de pénalisation des correspondances marche à pied.

Un label est étendu de manière différente en fonction du type d'arc par lequel il est étendu.

- Arc GETIN

$$\begin{cases} t' = & \mathit{prochainHoraire}(t + P_t, e) \\ C' = & \mathit{prochaineCourse}(t + P_t, e) \\ c' = & c + (t' - t) + P_c \end{cases}$$

- Arc ITI

$$\begin{cases} t' = & \mathit{horaire}(C, e) \\ C' = & C \\ c' = & c + (t' - t) \end{cases}$$

- Arc CORRESPWALK

$$\begin{cases} t' = & t + d \\ C' = & \emptyset \\ c' = & c + d \cdot C_w \end{cases}$$

- Arc GETOFF

$$\begin{cases} t' = & t + P_t \\ c' = & c + P_t + P_c \\ C' = & \emptyset \end{cases}$$

Enfin, un certain nombre de facteurs entrent également en compte lors de l'extension des labels et rendent certaines évaluations impossibles : la fonction ExtendLabel renvoie alors un coût égal à MAXEVAL et le label ne sera alors pas étendu. En voici la liste exhaustive. Certains rejets sont liés à des paramètres de la requête, tels qu'une nécessité d'accessibilité, une liste de modes, d'opérateur, d'arrêts ou de lignes interdits, une distance de correspondance marche à pied maximale.

- Pour tout type d'arc :
  - ▷ arrêt interdit dans la requête,
  - ▷ accessibilité de l'arrêt du nœud e incompatible avec la requête,
  - ▷ perturbation à l'arrêt du nœud e.
- Arc GETIN :
  - ▷ mode interdit dans la requête,
  - ▷ opérateur interdit dans la requête,
  - ▷ ligne interdite dans la requête,
  - ▷ aucun arc de sortie sur le nœud e,
  - ▷ erreur de donnée sur le nœud e,
  - ▷ accessibilité de la ligne incompatible avec la requête,
  - ▷ aucun prochain horaire trouvé,
  - ▷ erreur d'horaire incohérent ($h' < h$).
- Arc ITI :
  - ▷ erreur de données : pas d'horaire trouvé,
  - ▷ erreur de donnée : course actuelle.
- Arc CORRESPWALK :
  - ▷ marche à pied plus longue que la limite imposée,
  - ▷ marche à pied interdite en début ou fin de calcul,
  - ▷ interdiction de correspondances multiples,
  - ▷ erreur de donnée de temps ou distance de correspondance absurde.
- Arc GETOFF :
  - ▷ arc de descente interdit si l'arc évalué précédemment n'est pas un arc ITI.

Notons que certaines de ces règles, comme l'interdiction de correspondances mul-

tiples, rendent l'algorithme sous-optimal.

Règle de dominance

La dominance entre labels est extrêmement simple : pour deux labels L_1 de coût $c1$ et L_2 de coût $c2$, $L_1 \prec L_2 \Leftrightarrow c1 < c2$.

Règle de priorité

Les nœuds insérés dans le tas binaire sont triés par ordre croissant des coûts c de leurs labels.

Label de départ

Le label de départ est créé sur le nœud MARCHE correspondant à l'arrêt de départ A, avec les valeurs de ressources suivantes :

$$\begin{cases} c = 0 \\ t = \tau \\ C = \emptyset. \end{cases}$$

Condition d'arrêt et complexité

L'algorithme s'arrête lorsque le tas binaire est vide, ce qui arrive assez rapidement après avoir trouvé la première solution possible, puisqu'aucun label de coût supérieur à celle-ci n'est plus rajouté dans le tas binaire.

La complexité au pire cas de l'algorithme de Dijkstra est en $\mathcal{O}(m+n)\log(n)$, pour un graphe à m arcs et n nœuds. Vu que les graphes de transport en commun sont creux, avec environ cinq fois plus d'arcs que de nœuds, on a $m = \mathcal{O}(n)$, et donc une complexité en $\mathcal{O}(n)\log(n)$.

3 Développement d'un algorithme multilabel

Cette section traite de notre premier apport personnel dans ces travaux de thèse : l'évolution du calculateur de Cityway vers une version multilabel, basée sur l'algorithme de Martins. Nous détaillons ici beaucoup d'aspects techniques de ces modifications, car l'une des facettes de nos travaux est la complexité de l'intégration d'algorithmes théoriques à des produits existants. Il est important que les résultats soient identiques ou meilleurs que les résultats de l'algorithme précédent, tout en respectant les contraintes issues des clauses techniques industrielles.

Un calcul multicritère et multilabel nécessitent la définition :

- des ressources du label, un ensemble d'éléments définissant le label sur lesquels est optimisé ou contraint le calcul,
- de la règle d'extension des labels, c'est-à-dire de la fonction f_e qui à tout label L sur un nœud u et arc $l(u, v)$ du graphe associe un nouveau label $f_e(L)$,
- de la fonction de dominance définie par l'ordre partiel $\prec$ des labels sur le graphe
- d'une fonction scalaire de **coût** c qui associe un élément de $\mathbb{N}$ à tout label, ce qui permet de trier le tas binaire.

Les changements principaux pour implémenter cet algorithme sont d'une part la création de classes adaptées de labels et de coûts, et d'autre part la refonte du tas binaire pour qu'il contiennent des références vers des labels et non plus des nœuds.

Lors du début du calcul, une liste vide globale *allLabels* de labels est créée pour gérer la mémoire lors de la création de labels, et leur suppression à l'initialisation d'un nouveau calcul.

À chaque nœud n est associée la liste de ses labels n.Labels. Cette liste est ajustée lors de l'extension d'un label sur le nœud ainsi que décrit par l'algorithme 4.

Ainsi les labels sont présents dans une à trois collections distinctes : la liste du nœud correspondant, la liste globale des labels créés, et le tas binaire.

La fonction TryAddNewLabel(n, L), décrite par l'algorithme 3 vérifie s'il faut ajouter un label étendu L sur le nœud n. S'il existe au moins un label dans n.Labels qui domine strictement ou est équivalent L, rien n'est fait et on sort de la fonction. Sinon, tout label de n.Labels dominé par L est marqué comme dominé, et supprimé de la liste n.Labels. Enfin le nouveau label est ajouté aux listes {allLabels} et n.Labels.

Précisons que tout label supprimé par dominance reste dans le tas binaire et sera éliminé plus tard, lorsqu'il remontera jusqu'au nœud racine du tas.

Algorithm 3 TryAddNewLabel(n, L)

```
for all L_n ∈ n.Labels do
  if L_n ≺ L or L_n ≡ L then
    {L_n domine L ou lui est équivalent, aucune action.}
    return false
  else if L ≺ L_n then
    {L_n est dominé par L. On le supprime de la liste et on le flag comme dominé.}
    L_n.isDominated ← true
    n.Labels.Remove(L_n)
  end if
end for
n.Labels.Add(L)
allLabels.Add(L)
return true
```

Algorithm 4 Algorithme multilabel avec file de priorité

```
result ← nouvelle liste vide
allLabels ← nouvelle liste vide
{Mise en place du départs}
L_A ← CreateStartLabel(A).
Heap.Insert(L_A).
allLabels.Add(L_A)
{Boucle principale}
while !Heap.IsEmpty() do
  label ← Heap.DelMin()
  {On ne traite pas les labels dominés}
  if label.isDominated then
    continue
  end if
  {Extension des labels}
  for all outlink ∈ node.Outlinks do
    endLabel ← ExtendLabel(label, outlink)
    if endLabel.feasible() then
      if TryAddNewLabel(outlink.EndNode, endLabel) then
        Heap.Insert(endLabel)
      end if
      {Nœud d'arrivée : rajouter à la liste de résultats}
      if outlink.EndNode∈ Arrivées then
        result.Add(endLabel)
      end if
    end if
  end for
end while
{Purge des résultats dominés par d'autres}
for all label L ∈ result do
  for all label L_1 ∈ result do
    if L_1 ≺ L then
      result.Remove(L)
      continue
    end if
  end for
end for
return result
```

4 Algorithme bicritère d'arrivée au plus tôt et moins de changements

L'algorithme multilabel décrit précédemment est générique et s'adapte pour tout type de règle de dominance.

Cette section traite du calcul bicritère **arrivée au plus tôt** et **moins de changements**, qui doit renvoyer le trajet arrivant le plus tôt sans changement, le trajet arrivant le plus tôt avec au plus un changement, le trajet arrivant le plus tôt avec au plus deux changements, et ainsi de suite. Nous définissons les ressources, les règles d'extension et de dominance à appliquer au code multilabel pour obtenir ces résultats.

Un deuxième apport de la thèse a donc été de coder ces règles, au sein du cadre multilabel préalablement établi.

Nous détaillons ensuite les expérimentations sur l'algorithme bicritère « plus rapide et moins de changements » effectuées pour vérifier notre code sur les réseaux de transport d'une grande région métropolitaine R1, d'une région française R2 et d'une petite ville R3.

4.1 Description de l'algorithme

Nous décrivons dans cette section les ressources d'un label, ainsi les règles de dominance, d'extension, d'initialisation des labels et de priorité du tas utilisées pour l'algorithme bicritère.

Ressources

L'algorithme bicritère cherche à optimiser deux critères, le nombre de courses empruntées et le temps écoulé depuis l'horaire de départ demandé :

- nombre de courses c : égal au nombre d'arcs de type GETIN dans le trajet. Si l'on souhaitait privilégier certains modes par rapport à d'autres, on pourrait définir un coût de changement par mode. c serait alors égal à la somme de tous les coûts de changement de tous les arcs GETIN du label.
- durée o : égale au temps écoulé depuis l'horaire de départ τ. Il serait aisé de pénaliser divers éléments du label et transformer cette ressource en somme pondérée similaire au « coût » de l'algorithme monolabel. Par exemple une pénalisation de la marche à pied permettrait de favoriser le « moins de marche à pied ».

Les autres attributs utilisés pour le calcul sont les mêmes qu'en monolabel, à savoir :

- un temps t, correspondant à l'horaire actuel,
- la dernière course C empruntée par ce label.

Règle de dominance

La règle de dominance est celle de la définition 31 avec les deux critères (c, o) : nombre de courses et durée du label. On peut la traduire par la formule suivante.
$L_1(c_1, o_1) \prec L_2(c_2, o_2) \Leftrightarrow (c_1 < c_2 \wedge o_1 \leq o_2) \vee (c_1 \leq c_2 \wedge o_1 < o_2)$.

Règle d'extension

Les règles d'extension sur les ressources t et C, ainsi que les conditions de réalisabilité sont identiques au cas monolabel décrit dans la section correspondante. Nous ne détaillerons donc pas à nouveau ces calculs. Les ressources c et o sont mises à jour de la manière suivante.
Pour un label $L(c, o)$ et un arc a, $L'(c', o')$ est défini par

$$c' = \begin{cases} c + 1 & \text{si } a \text{ est de type GETIN} \\ c & \text{sinon} \end{cases}$$
$$o' = t - \tau$$

Règle de priorité

L'utilisation d'un tas binaire nécessite la définition d'une règle de priorité, c'est-à-dire d'une relation d'ordre total entre les labels, que l'on exprimera par une fonction des attributs de label vers $\mathbb{R}$.

La règle d'agrégation proposée est de la forme $\lambda c + o$, avec lambda de l'ordre de 10000. Cela permet de calculer les labels dans l'ordre du nombre de changements, et de minimiser le nombre d'extensions d'arc de montée, plus couteuses car nécessitant le calcul du prochain horaire à l'arrêt.

Labels de départ

Comme pour le monolabel, le labels de départ est créé sur le nœud MARCHE correspondant à l'arrêt de départ de la requête A, avec les valeurs de ressources suivantes :

$$\begin{cases} c = 0 \\ o = 0 \\ t = \tau \\ C = \emptyset. \end{cases}$$

Il est possible lors d'un calcul avec départs multiples d'initialiser o et t avec le temps d'accès à ces points de départ . Ceci est particulièrement utile pour les calculs combinés avec rabattement sur un graphe piéton, vélo ou voiture.

Condition d'arrêt et complexité

L'algorithme s'arrête lorsque le tas binaire est vide.

Dans le cas général, un tel algorithme est théoriquement NP-difficile. En pratique, il dépend de la règle de dominance utilisée. Ici, il y a une très forte et évidente corrélation entre le nombre de changements et la durée des labels. On peut donc anticiper la présence d'un nombre très limité de labels par nœud : en effet il n'y a pour un nœud qu'au maximum un label par nombre de changements possible ; or, il est très rare, même dans des réseaux combinant les transports de toute une région, d'avoir des solutions optimales entre deux arrêts de plus de cinq changements. Nous montrons dans les sections suivantes que les temps d'exécution de l'algorithme multilabel sont en moyenne deux fois plus longs que ceux du monolabel.

4.2 Instances d'expérimentations

Nous avons testé l'algorithme multilabel sur les réseaux R1, R2 et R3, très différents par leur topologie de transport et les quantités de données traitées, respectivement les réseaux d'une métropole nord-américaine, d'une grande région française et d'une petite ville française. Nous ne nous intéresserons pas à la taille des graphes « marche, vélo et voiture », car les expérimentations effectuées ne portent que sur des trajets en transport en commun uniquement, sans rabattement.

Réseau R1

Le réseau le plus important et le plus dense de nos expérimentations est celui-ci. Il comprend plus de dix exploitants, dont un gigantesque réseau de bus de ville très bien fourni et un réseau de cars et trains reliant les villes environnantes. Le graphe TC comprend environ 170k nœuds et plus de 37k arrêts. Mille trois cent lignes et 365k courses parcourent ce réseau pour plus de 2.7M d'horaires à l'arrêt par jour. De plus, le client a choisi d'avoir un rayon de correspondance marche à pied de 500 mètres, ce qui induit plus de 140k correspondances entre arrêts physiques.

Réseau R2

L'autre réseau de test de grande taille est celui d'une grande région française. Il comprend plus de quarante exploitants, dont les plus gros sont des réseaux de bus urbains et des réseaux de cars départementaux. Le graphe TC comprend environ 174k nœuds pour plus de 27k arrêts. Trois mille cent lignes et 60k courses parcourent ce réseau pour environ 650k horaires à l'arrêt par jour. De plus, le réseau comprend 91k correspondances.

Réseau R3

Le troisième réseau étudié est un très petit réseau urbain, utilisé principalement dans l'optique de tester les non régressions et le comportement de l'algorithme multilabel sur un petit graphe. Il n'y a qu'un seul partenaire de bus, et 3600 nœuds dans le graphe pour 700 arrêts. Il y a environ 14k horaires par jour, avec 1800 courses et 55 lignes. Enfin il y a 580 correspondances.

Réseau	Nœuds	Arcs	Horaires
R1	170 000	680 000	2 700 000
R2	175 000	608 000	650 000
R3	3 000	9 000	14 000

Tableau 5 – Taille des instances de test

4.3 Expérimentations

Nous présentons dans cette section les expérimentations effectuées et leurs résultats. De plus, nous exploitons certains résultats pour analyser les différents réseaux de transport en commun testés.

Protocole de tests

Nous avons développé un générateur de requêtes d'itinéraire aléatoires en se basant sur les jeux de fichiers décrivant les réseaux. Nous avons ensuite généré 1000 requêtes par réseau, avec un arrêt de départ aléatoire, un arrêt d'arrivée aléatoire et un horaire aléatoire compris entre 07:00 et 09:00 pour le jeu de données heures pleines, ou 09:00 et 16:00 pour le jeu de données heures creuses. Chaque requête est en mode «départ à», sans filtrage de mode, d'arrêt de lignes ni aucune autre subtilité. Les rabattements en marche à pied en début ou fin de calcul sont interdits, sauf par le biais de correspondances marche à pied, inclues dans le graphe TC.

Les calculateurs ont été configurés dans leur version monolabel comme dans leur version multilabel sans pénalité d'aucune sorte pour la marche à pied ni pour le nombre de changements ($\mathsf{C}_w = \mathsf{1}$ et $\mathsf{P}_c(\mathsf{M}) = \mathsf{0}$).

Les tests ont été effectués sur un ordinateur portable Dell avec un processeur Intel Core i7-3632QM cadencé à 2.2GHz, sous Windows 7 Professionnel.

Nous appelons v6 la version du calculateur correspondant à l'algorithme de Dijkstra avant modification. Nous appelons v7 la version modifiée pour être utilisable en multilabel, avec les distinctions v7 monolabel et v7 multilabel pour signifier que c'est le même code, correspondant à l'algorithme 4 mais des règles de dominance différentes. V7 monolabel indique que la règle de dominance ne se fait que sur un critère, la durée, et les résultats devraient être les mêmes que ceux de la v6. V7 multilabel indique un calcul bicritère temps/nombre de changements.

Résultats multilabel

Nous avons tout d'abord vérifié l'équivalence des résultats entre l'ancienne version du moteur (v6) et la version multilabel (v7) utilisée en monolabel. Ces tests ont été utiles pour développer une version du calculateur sans régression, à la fois en ce qui concerne les temps de calculs, qui doivent être similaires, et les réponses, qui doivent être identiques.

La figure 23 établit la répartition des temps de calcul v7 monolabel et v7 multilabel en fonction du temps de calcul de la v6, pour le réseau de R1 en heures de pointe.

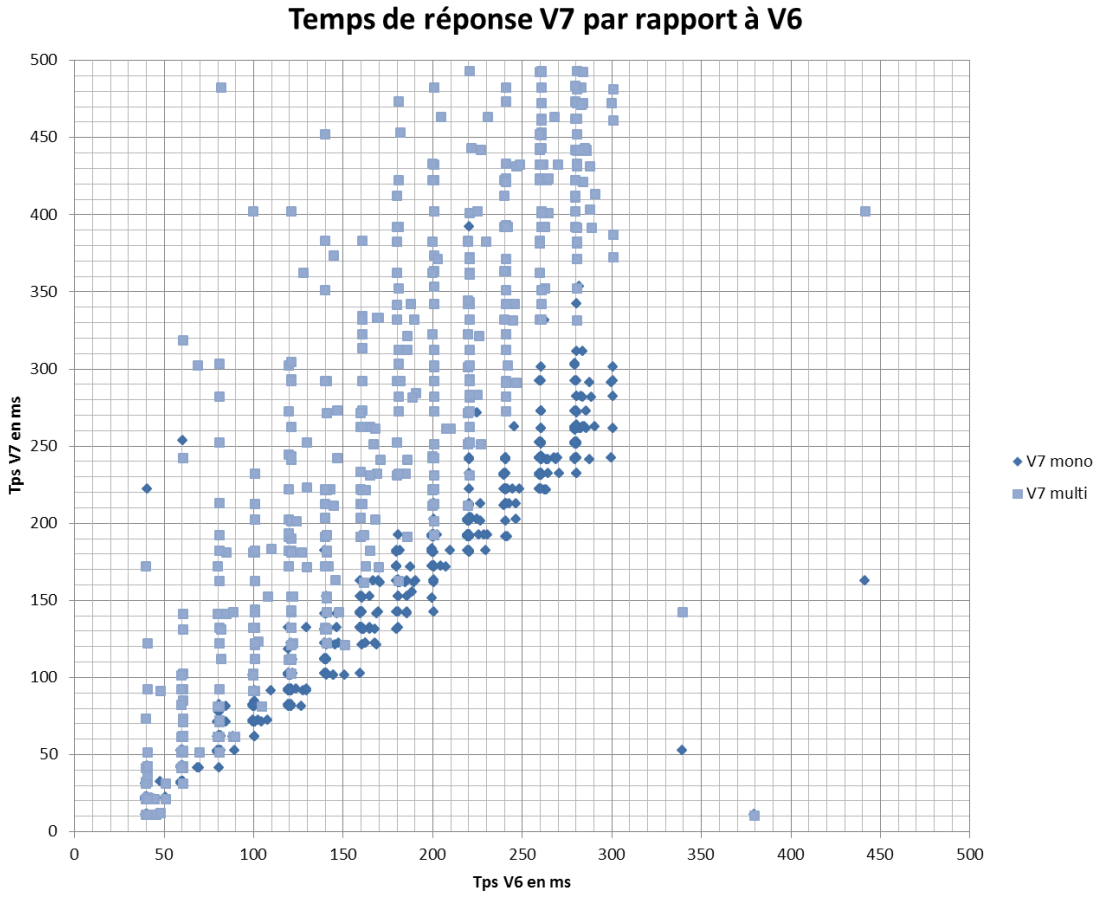


FIGURE 23 – Temps d'exécution pour R1

Les tests montrent que les temps de réponses entre la version v6 et la version v7 monolabel sont très proches. Le graphe 23 le montre sur le réseau de R1 en heures de pointe, et les résultats sur les autres réseaux mènent à la même conclusion. De plus, 100 % des solutions sont identiques entre les deux versions pour les six jeux d'essais.

Lors du passage à l'algorithme multilabel, le tas binaire a été modifié et l'opération Decrease n'est plus effectuée. Les deux opérations Insert et Delete utilisées à la place sont potentiellement plus couteuses en temps CPU. De plus, l'utilisation de labels créés à la volée et détruits en fin de calcul aurait également pu nuire aux performances, comparés au code v6 dans lequel tous les attributs et ressources du nœud sont pré-alloués en mémoire. Finalement, ces considérations théoriques n'ont pas eu d'impact en

pratique.

Le tableau 4.3 affiche les temps moyens et maximums d'exécution sur les jeux de requêtes en heure de pointe.

Réseau	v7 mono moy.	v7 multi moy.	v7 mono max.	v7 multi max.
R1	150	291	392	764
R2	106	164	311	486
R3	12	11	95	31

Tableau 6 – Temps d'exécution des requêtes de test

Sur R3, les moyennes de temps de calcul sont très proches du minimum possible. Le calcul se termine ici quasi instantanément et l'information n'est donc pas pertinente.

Sur les jeux de données les plus volumineux, nous constatons une augmentation des temps de calculs de 50 % à 100 % environ du v7 multi par rapport au v7 mono. Nous voyons sur la figure 23 qu'il y a une assez forte dispersion, certaines requêtes prenant jusqu'à cinq fois plus de temps en multilabel, ce qui rend bien compte de la difficulté à prédire comment le nombre de labels par nœud évolue.

Nous constatons donc une augmentation globale du temps de réponse en multilabel assez marquée. Cette augmentation reste toutefois acceptable puisque les réponses donnent un ensemble de solutions, et non une solution unique. De plus, les temps de réponses maximum restent raisonnables pour un utilisateur final.

Une campagne d'essai supplémentaire a été effectué en v6 monolabel avec le critère de « moins de changements », qui pénalise lourdement les changements. Les résultats sont inclus pour 97 % dans les solutions obtenue avec la version v7 multilabel ; les 3 % restants sont meilleurs en multilabel, pour des raisons d'écrasement de labels en Dijsktra classique monolabel.

Nombre de solutions par réseaux

La figure 24 montre le nombre de solutions par requête et concerne les instances de test de R1, R2 et R3 en heure pleines.

Le réseau R1 est exceptionnellement dense et bien pensé, puisqu'il donne une ou plusieurs solutions dans 97 % des cas, contrairement au réseau de R2, dans lequel seulement 60 % des requêtes aléatoires aboutissent. Cela donne une idée de la configuration de R2, en particulier du fait qu'un grand nombre d'arrêts du réseau sont inaccessibles à certains horaires de la journée, ou jour de la semaine, si l'origine du trajet est trop éloignée. Sur R3, 17 % des requêtes n'ont pas de solutions, ce qui est relativement élevé pour un réseau mono-transporteur : chaque arrêt pourrait être accessible à tous les moments de la journée.

Cette statistique fournit aussi une information sur l'apport du multilabel en terme de nombre de solutions par requête, lorsqu'il y en a au moins une. Pour R3, il n'y a qu'une seule solution dans 80 % des cas, deux solutions dans 20 % des requêtes, le

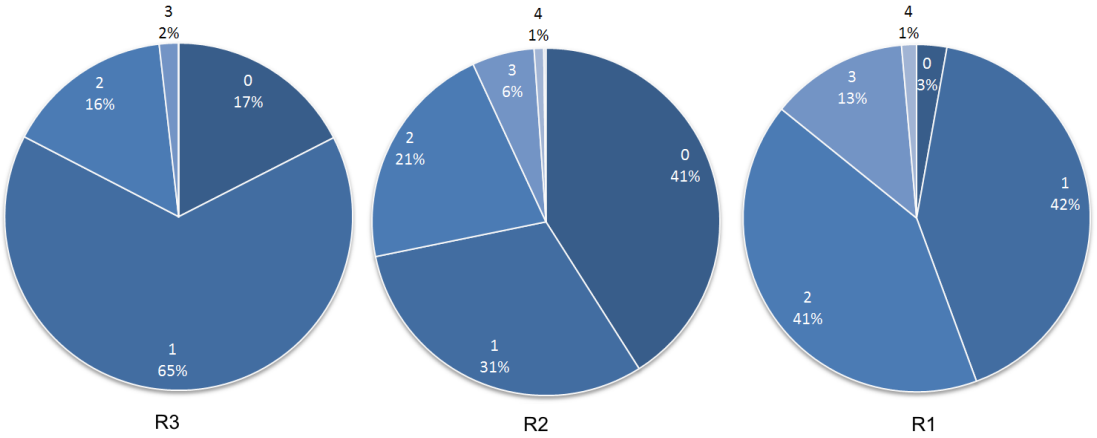


FIGURE 24 – Répartition du nombre de solutions (R3, R2, R1)

multilabel n'apporte donc que peu d'alternatives. En revanche, pour R1 et R2, 35 % à 40 % des solutions offrent deux alternatives, et environ 10 % à 15 % des solutions en offrent trois ou plus. Cela justifie et valide l'idée de chercher un front de Pareto plutôt qu'une solution unique.

Les résultats des requêtes en heures creuses sont sensiblement identiques, à ceci près que la proportion de requêtes sans solutions est plus grande (6 % pour R1, 61 % pour R2 et 27 % pour R3).

Nombre de courses par solution

La figure 25 indique la répartition des trajets obtenus en fonction du nombre de courses de chacun d'eux, pour chaque instance de test en heures pleines.

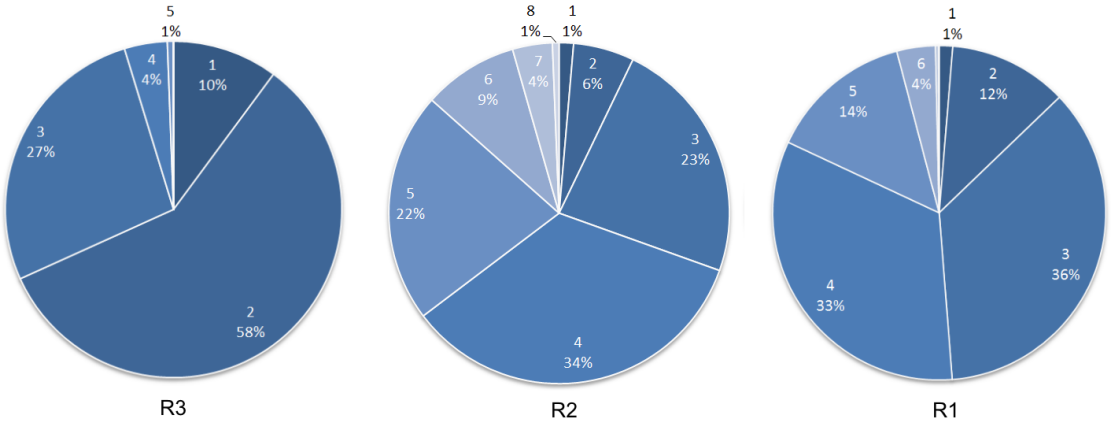


FIGURE 25 – Répartition du nombre de courses empruntées (R3, R2, R1)

Notons que l'on parle indifféremment du nombre de courses d'un trajet ou du nombre de changements d'un trajet, le premier étant égal au second plus un.

Cette statistique décrit à la fois la taille géographique du réseau et la qualité de son graphicage : en effet, un réseau dans lequel les trajets optimaux entre des arrêts

aléatoires comprennent peu de changements est assez petit pour qu'un faible nombre de lignes couvrent potentiellement tous les arrêts et bien optimisé pour que ce soit le cas en pratique. En revanche, un grand réseau, ou un réseau moins bien pensé aura des trajets optimaux avec plus de changements en moyenne.

De plus, cela donne également une idée du nombre de changements qu'un utilisateur du calculateur d'itinéraire peut s'attendre à avoir dans sa réponse.

Ici, on constate sur R3 que la plupart des solutions ont un, deux ou aucun changements, dans l'ordre de fréquence. Sur R2, les solutions ont le plus souvent deux, trois et même quatre changements dans 22 % des cas. Le réseau est donc à la fois grand, ce que l'on savait déjà, et aussi plus fragmenté : et il est difficile d'aller d'un bout à l'autre de la région en transport en commun.

Sur R1, les résultats sont proches de ceux de R2, avec tout de même beaucoup plus de trajets optimaux en seulement deux changements. Le réseau est également grand, mais plus dense : il est moins difficile de trouver un trajet optimal avec peu de changements.

Temps gagné au sacrifice de changements

Enfin, les figures 26, 27, et 28 fournissent une information statistique sur le temps gagné en moyenne au sacrifice d'un ou plusieurs changements, pour les jeux de test R1 et R2 en heures pleines. Définissons ce que l'on entend par « temps gagné au sacrifice d'un changement ». Chaque réponse multilabel comprend une ou plusieurs solutions $\mathcal{S}_i = S@\tau \xrightarrow[n_i]{} A@\tau_i$ avec les séries n_i strictement décroissante, et τ_i strictement croissante. Pour chaque couple de solutions s'écrivant $(\mathcal{S}_i, \mathcal{S}_{i+1})$, appartenant à la même réponse, nous définissons le gain de temps au sacrifice d'un ou plusieurs changements égal à $\tau_{i+1} - \tau_i$. Nous comptons ensuite les occurrences de ces gains de temps pour toutes les solutions et nous les regroupons par tranche de cinq minutes pour créer les histogrammes.

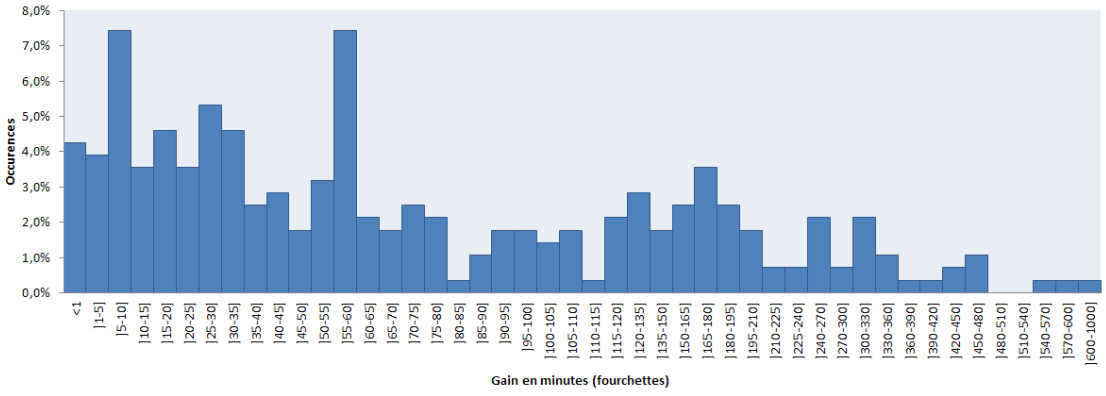


FIGURE 26 – Fréquence du temps gagné au sacrifice de changements (R2)

Note : ces mêmes données ont été obtenues pour les jeux de données en heures creuses, mais nous les omettons afin de ne pas surcharger la section, étant donné que les résultats ne sont pas très différents de ceux présentés et n'apportent pas plus d'informations.

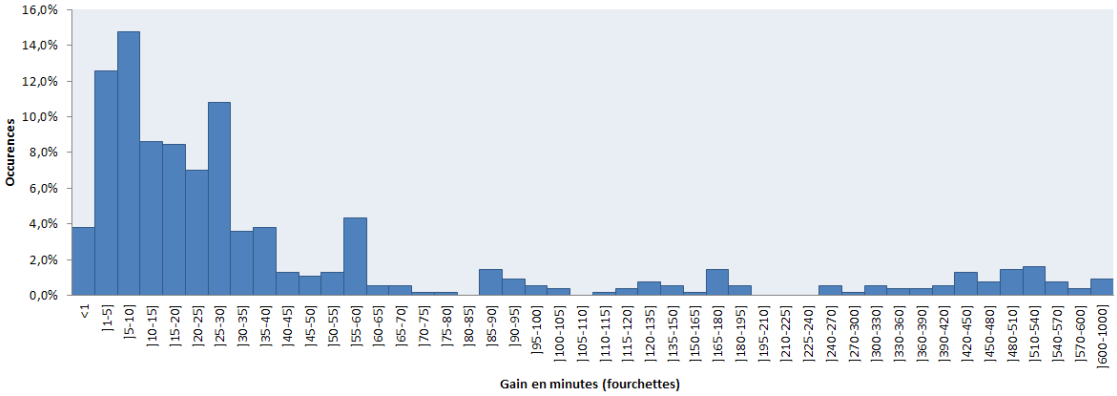


FIGURE 27 – Fréquence du temps gagné au sacrifice de changements (R1)

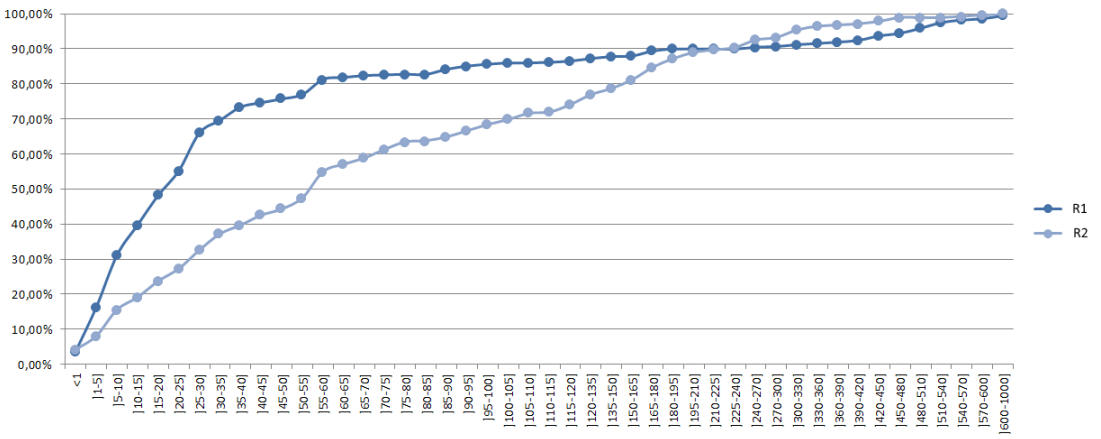


FIGURE 28 – Fréquence cumulée du temps gagné au sacrifice de changements

Cette statistique est très intéressante du point de vue de l'analyse d'une offre de transport en commun. Elle permet de visualiser avec précision le temps perdu en moyenne, lorsque l'on choisit d'économiser un changement, ou, de manière équivalente, le temps gagné en moyenne lorsque l'on choisit de sacrifier un ou plusieurs changements.

Pour R2, nous constatons en regardant la figure 26 que dans la majorité des cas, sacrifier un changement fait gagner entre une et soixante minutes sur le trajet. Cependant, il y a également beaucoup de cas où le temps gagné est entre 1h et 4h. Cela signifie que même lorsqu'il existe des trajets assez directs, leurs faibles fréquences de circulation font qu'il sera souvent nécessaire de réaliser plus de changements pour ne pas perdre plusieurs heures.

Pour R1, nous constatons en regardant la figure 27 que dans la majorité des cas, sacrifier un changement fait gagner entre une et trente minutes, avec très peu d'occurrences au-delà. Cela signifie que même si prendre un transport supplémentaire peut faire gagner un peu de temps, choisir un trajet plus lent avec moins de changements est le plus souvent une solution intéressante.

La figure 28 reprend ces arguments en comparant les courbes cumulées de ces nombres. Pour R1, environ 70 % des trajets économisent moins d'une demi-heure en sacrifiant un changement, alors que pour R2 seulement 38 % des trajets économisent moins d'une demi-heure en sacrifiant un changement.

5 Conclusion

En conclusion, nous pouvons dire que le changement de l'algorithme du monolabel au multilabel est un succès du point de vue des expérimentations sur le cas bicritère : les solutions renvoyées sont variées, pertinentes, et répondent aux exigences que nous nous sommes imposées. Les temps de calcul, bien que plus élevés, restent raisonnables. Dans une optique de mise en production, ils pourront être améliorés par des optimisations heuristiques, telle que la limitation des labels dès qu'une solution a été calculée, en s'aidant d'un ordre de tas binaire bien choisi. De plus, il est tout à fait envisageable de dupliquer des instances de calculateurs et de créer des IHM qui font des appels asynchrones d'une part sur le calculateur monolabel pour un premier résultat rapide, et d'autre part sur le calculateur multilabel pour un résultat exhaustif.

Notons également qu'un calculateur d'itinéraire véritablement multicritère peut servir de base à l'analyse des réseaux de transports en commun et permettre de générer divers graphiques (nombre de changements des trajets optimaux, temps gagné au sacrifice d'un changement, nombre de solutions optimales ...).

Enfin, cet algorithme multilabel peut être étendu à d'autres ressources et règles de dominance, apportant à Cityway des possibilités de développement à court terme. Par exemple, nous pouvons l'exploiter pour prendre en compte des critères supplémentaires comme le trajet le moins cher, ou le plus fiable. De plus, la possibilité d'avoir plusieurs labels par nœud permet également de développer des règles résolvant les problèmes liés à l'écrasement de labels intéressants, décrits dans la section 4 du chapitre III, tout en conservant la souplesse et les contraintes prises en charge par le code actuel.

Chapitre V

Génération et sélection d'un ensemble diversifié de trajets alternatifs

Sommaire

Ce chapitre, consacré à l'élaboration d'ensembles de trajets diversifiés, est organisé de la manière suivante. Après une brève introduction, les sections 2 et 3 présentent respectivement l'algorithme monolabel et l'algorithme multilabel permettant de générer des grands ensembles de solutions. La section 4 détaille les algorithmes de sélection des trajets à renvoyer parmi les solutions générées. La section 5 complète le chapitre avec des expérimentations numériques qui évaluent les algorithmes proposés.

1 Introduction

Dans un réseau de transport dense et bien structuré, comptant plusieurs dizaines d'exploitants, comme celui de R1 (cf. chapitre IV), il existe de nombreuses alternatives de qualité à la solution la plus rapide, ou à la solution ayant le moins de changements. Les critères de préférences d'un utilisateur sont de plus dans la majorité des cas plus complexes que ces critères simples. Par exemple l'utilisateur ne souhaitera pas prendre le métro, si le temps de transport additionnel reste limité, ou il privilégiera un bus qui traverse de beaux quartiers. Une solution dominée sur des critères simples, comme l'horaire d'arrivée ou le nombre de courses peut ainsi se révéler préférable pour l'utilisateur final.

Actuellement, le calculateur d'itinéraire de Cityway répond à une requête de l'utilisateur par une feuille de synthèse de trois trajets en transport en commun. Ceux-ci sont obtenus par l'algorithme monolabel, répété successivement à des horaires légèrement décalés. Pour cette raison, cette feuille de synthèse ne propose qu'une pseudo-diversité des résultats, avec le plus souvent trois solutions répétant le même schéma de lignes.

Permettre à l'utilisateur de faire varier sa requête, en interdisant manuellement modes, opérateurs, lignes ou arrêts crée des solutions alternatives et peut permettre de trouver la solution préférée de l'utilisateur. Ce dernier se voit cependant contraint de relancer plusieurs fois sa requête, jusqu'à converger, par tâtonnement, vers la solution la plus adaptée à ses besoins. L'objectif de ce chapitre est de développer des méthodes pour maximiser les chances de satisfaire au mieux l'utilisateur, le tout sans instructions manuelles (afin de minimiser le nombre d'opérations effectuées par l'utilisateur final) et donc sans connaissance sur les préférences de l'utilisateur. Pour cela, nous proposons de générer un ensemble diversifié de bonnes solutions, parmi lesquelles l'utilisateur pourra faire son choix.

La mise en place de telles méthodes soulève plusieurs difficultés. Une première difficulté est de trouver la solution préférée de l'utilisateur sans connaître ses préférences ! C'est pour y faire face que nous proposons de générer un ensemble de solutions diverses, maximisant ainsi les chances de « bien tomber ». Une deuxième difficulté est alors de modéliser la notion de diversité. Une troisième difficulté est d'ordre algorithmique : comment générer rapidement un ensemble de solutions de qualité acceptable selon les critères classiques et présentant une bonne diversité ? La dernière difficulté concerne enfin la taille de l'ensemble de solutions retourné à l'utilisateur : combien retenir de solutions et comment arbitrer entre qualité et diversité ?

Pour faire face à ces difficultés, nous avons découpé notre approche en deux étapes principales. Dans un premier temps, nous nous attachons à obtenir un grand nombre de solutions réalisables de bonne qualité en un temps raisonnable. Dans un second temps, nous extrayons un nombre raisonnable de solutions diversifiées de cet ensemble. La première étape est réalisée selon deux variantes : un algorithme basé sur la répétition de l'algorithme monolabel, détaillé dans la section 2, ou une variante de l'algorithme multilabel, décrite dans la section 3. Pour la deuxième étape, nous avons utilisé plusieurs algorithmes de sélection des solutions, introduits dans la section 4. Notons que ces algorithmes, bien qu'initialement destinés à choisir des solutions en faisant un compromis entre qualité et diversité, pourraient également servir à choisir des solutions dans un front de Pareto d'une requête multicritère aux critères bien définis.

Tout au long de ce chapitre, nous ne nous intéressons pas aux trajets comprenant des portions en vélo ou en véhicule privé. Bien que ce soit un problème similaire, et même si les travaux de ce chapitre pourrait s'appliquer à de tels trajets, les durées de rabattement vélo et voiture sont plus longs, ce qui complexifie le problème et pose de nouvelles questions, comme l'importance de la variabilité de ces durées sur la diversité. Nous traiterons donc ces cas dans un temps ultérieur aux travaux de thèse, lorsque les méthodes présentées ici auront été validées industriellement.

Les algorithmes utilisés dans ce chapitre, que ce soit en monolabel ou multilabel, optimisent les trajets trouvés au sens de la somme pondérée qui agrège durée, nombre de changements et durée de marche à pied. Contrairement aux expérimentations du

chapitre IV, les expérimentations de ce chapitre autorisent les rabattement en marche à pied en début et fin de trajet.

2 Génération de trajets avec l'algorithme monolabel

Les algorithmes présentés dans cette section consistent en la résolution successive d'algorithmes monolabels. Le principe général est inspiré par les travaux de Yen [Yen1971] sur les calculs de K plus courts chemins, que nous présentons ci-dessous avant de présenter nos algorithmes.

2.1 Principe de l'algorithme des K plus courts chemins

Nous détaillons d'abord le fonctionnement d'un algorithme de calcul des K plus courts chemins dans un graphe, dont l'idée générale sera reprise dans nos algorithmes. Le principe est de trouver d'abord le trajet optimal en appliquant l'algorithme monolabel classique, puis le deuxième meilleur trajet, le troisième et ainsi de suite jusqu'au $K^{\text{ième}}$.

A chaque étape du calcul, l'ensemble des trajets réalisables est partitionné pour interdire les trajets obtenus jusque là. Ces partitions sont basées sur les étapes du meilleur trajet.

Soit $T = (e_i)$ avec $i \in [1, k]$ le meilleur trajet, composé d'étapes élémentaires e_i. Soit T' un trajet réalisable différent de T. Alors, ou bien T' ne commence pas par la même étape élémentaire e_1, ou bien il existe $j \in [1, k-1]$ tel que T' commence par les étapes élémentaires $(e_1, \ldots, e_j)$ et son étape $j+1$ est différente de e_{j+1}.

Définition 34 *Nous noterons une portion de trajet ne commençant pas par l'étape élémentaire* e *avec le symbole* $\neg e$.

L'ensemble des trajets réalisables peut donc être représenté par l'union des ensembles disjoints suivants :

- le trajet T,
- l'ensemble des trajets s'écrivant $(\neg e_1)$,
- l'ensemble des trajets s'écrivant $(e_1, \neg e_2)$,
- l'ensemble des trajets s'écrivant $(e_1, e_2, \neg e_3)$,
- ...
- l'ensemble des trajets s'écrivant $(e_1, \ldots, e_{k-1}, \neg e_k)$.

Le meilleur trajet peut être calculé, indépendamment, pour chacun de ces ensembles (hormis le premier ensemble, réduit à T). Il suffit pour cela d'initialiser l'algorithme avec un label représentant le début du trajet et d'interdire l'extension de ce label vers l'étape interdite. Le deuxième meilleur trajet sera alors le meilleur des meilleurs trajets

obtenus. Le procédé sera répété pour obtenir le troisième meilleur trajet, et ainsi de suite. A chaque étape, un nouvelle série de calculs de plus court chemin devra donc être exécutée (précisons que le meilleur trajet résultant de l'étape i sera le meilleur des meilleurs trajets obtenus à l'étape i ou aux étapes précédentes, à l'exception bien sûr des $i-1$ meilleurs trajets).

Dans notre contexte, le nombre de calcul à réaliser est bien trop élevé pour utiliser telle quelle cette méthode. De plus, rien ne favorise la diversité des solutions et un risque important existe de générer des solutions quasiment similaires. Par exemple, si deux bus suivent une portion de trajet commune sur K arrêts et que la meilleure solution nécessite une correspondance entre ces bus à n'importe lequel de ces arrêts, les K meilleures solutions seront en fait identiques au choix de l'arrêt de correspondance près.

Ci-après, nous allons proposer des heuristiques basées sur cet algorithme de calcul des K meilleurs chemins, dans le but simultané d'accélérer les temps de calcul et de favoriser la diversité.

2.2 Algorithme basé sur l'interdiction de courses

Une première modification consiste à éliminer de la représentation du trajet T tous les arcs de type CORRESPWALK, GETIN ou GETOFF. Rappelons en effet que pour aller d'un nœud ITI n_1 du trajet T au nœud ITI n_2 de l'itinéraire suivant du trajet T, au même arrêt ou en passant par une correspondance marche à pied, un seul chemin existe dans le graphe. Ce chemin est composé d'un arc de descente suivi éventuellement d'un arc de marche à pied, suivi d'un arc de montée. Chercher des alternatives à ces portions de trajet revient donc à modifier les itinéraires précédents ou suivants. Pour cette raison, partant de l'écriture $T = (e_i)$ avec $i \in [1, k]$ contenant toutes les étapes élémentaires du trajet, nous commençons par extraire la liste des étapes élémentaires de type itinéraire.

La deuxième modification concerne encore la représentation des trajets. Au lieu de considérer toutes les étapes élémentaires, nous ne retenons que les courses. Ainsi, par exemple, un bout de trajet en bus avec une descente située 15 arrêts après la montée, ne sera pas représenté par 15 étapes élémentaires (bouts de trajet entre deux arrêts), mais une seule (la course). En reprenant les notations introduites dans le chapitre I, définition 16, nous noterons $T = (c_1, \ldots, c_n)$. Cette simplification permet principalement de limiter le nombre de calculs à réaliser, mais est aussi intéressante pour favoriser la génération de trajets empruntant des lignes distinctes, et donc la diversité.

Une troisième modification est liée à des considérations techniques. L'implémentation ne permet pas, sans alourdissement conséquent du code, d'imposer le début d'un trajet. Dans notre algorithme, nous n'intégrons pas ces contraintes. Partant d'un trajet $T = (c_1, \ldots, c_n)$, l'ensemble des trajets réalisables sera donc représenté par l'union des ensembles :

- le trajet T,
- l'ensemble des trajets s'écrivant $(\neg c_1)$,
- l'ensemble des trajets s'écrivant $(\neg c_2)$,

- l'ensemble des trajets s'écrivant ($\neg c_3$),
- ...
- l'ensemble des trajets s'écrivant ($\neg c_n$).

Contrairement au cas usuel, les ensembles ne sont donc pas disjoints. Nous élargissons les ensembles de recherche, qui se recoupent, et il se peut ainsi que la même solution soit calculée plusieurs fois, auquel cas les occurrences supplémentaires ne sont pas prises en compte.

La quatrième modification est la suivante. Lorsqu'une course est interdite, il semble raisonnable d'interdire également les courses de même itinéraire planifiées peu de temps après. C'est particulièrement clair dans le cas du métro par exemple : il n'est pas souhaitable que l'interdiction d'une course en métro se traduise simplement par le fait d'attendre le métro suivant ! Pour limiter ce risque, lorsqu'une course est interdite, l'algorithme interdit également toutes les courses de même itinéraire planifiées dans les κ_1 minutes suivantes, où κ_1 est un paramètre.

Enfin, nous avons mis en place un système de filtre des solutions, afin d'une part de réduire les temps de calcul, mais aussi et surtout de ne pas présenter de trajets trop longs et comportant trop de changements par rapport au meilleur trajet. Ce filtre est basé sur un coefficient κ_2 appliqué au coût des labels : si le coût d'un label multiplié par κ_2 dépasse le coût du meilleur trajet, ce label est ignoré.

L'algorithme découle naturellement de ce qui a été expliqué précédemment (Algorithme 5). L'ensemble des « meilleures » solutions $\mathcal{S}$ est initialisé avec le meilleur trajet S_0, calculé avec l'algorithme monolabel classique. Une boucle permet, à chaque itération, de rajouter une nouvelle solution à $\mathcal{S}$ jusqu'à atteindre les K solutions recherchées. A la $k^{\text{ième}}$ itération, la $k^{\text{ième}}$ meilleure solution S est sélectionnée. Une méthode $f(S)$ effectue autant de calculs d'itinéraire qu'il y a de courses dans S, selon le principe expliqué précédemment, en rajoutant chacune d'elle aux courses interdites pour obtenir S. $f(S)$ renvoie les solutions obtenues, qui sont ajoutées à l'ensemble $\mathcal{S}_a$. Cet ensemble mémorise tous les résultats des calculs d'itinéraire effectués depuis la première itération. La meilleure solution de cet ensemble est la $(k+1)^{\text{ième}}$ meilleure solution. Cette solution est supprimé de $\mathcal{S}_a$, ajoutée à $\mathcal{S}$ et utilisée pour l'itération suivante.

Les K meilleures solutions sont alors dans l'ensemble $\mathcal{S}$, mais l'algorithme renvoie plutôt l'ensemble $\mathcal{S} \cup \mathcal{S}_a$, c'est-à-dire toutes les solutions calculées, pas seulement les meilleures. En effet, ce n'est qu'à l'étape de sélection que nous affinerons cet ensemble.

2.3 Illustration

Nous illustrons l'arbre de recherche parcouru par l'algorithme sur l'exemple de la figure 29, pour $K = 4$. La première solution S_0 calculée est composée de trois courses c_1, c_2, c_3. Trois calculs interdisant successivement ces trois courses sont donc lancés dans la boucle intérieure, permettant de trouver $f(S_0)$, composé de trois solutions. La meilleure solution alternative est notée S_1, trouvée en interdisant c_2. Elle est composée de c_1, c_4, c_3. Trois nouveaux calculs sont alors lancés pour construire $f(S_1)$, en interdisant successivement c_2, c_1, c_2, c_4 puis c_2, c_3. Les trois solutions de $f(S_1)$ sont rajoutées à

Algorithm 5 Algorithme de diversité monolabel

```
S ← S_0
𝒮.add(S)
𝒮_a ← ∅
while |𝒮| < K do
   𝒮_a ← 𝒮_a ∪ f(S)
   if 𝒮_a = ∅ then
      break
   end if
   S ← 𝒮_a.pop_best()
   𝒮.add(S)
end while
return 𝒮 ∪ 𝒮_a
```

l'ensemble des deux solutions restantes issues de $f(S_0)$, et S_2, la meilleure parmi les cinq, est sélectionnée. S_2 résulte de l'interdiction de c_1 et est composée de c_5, c_6. L'ensemble $f(S_2)$ est obtenu en interdisant d'une part c_1, c_5, d'autre part c_1, c_6. Ces deux solutions sont ajoutées aux quatre solutions restantes. La meilleure solution, S_3, est identifiée, ce qui conclut l'algorithme.

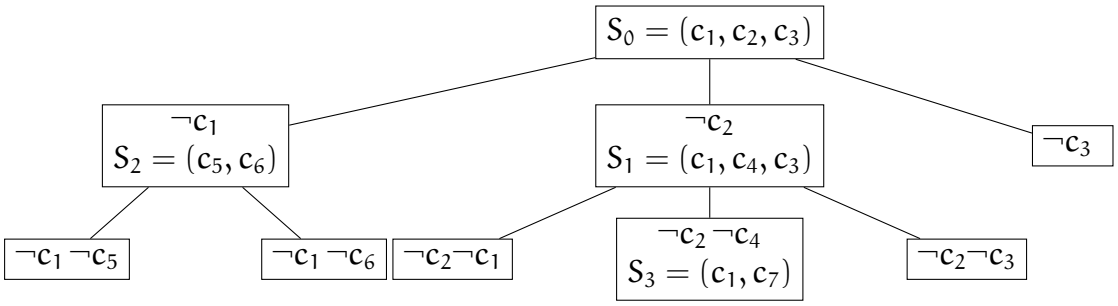


FIGURE 29 – Exemple d'arbre de recherche généré par l'algorithme 5

2.4 Optimisation

Comme expliqué précédemment, le découpage utilisé ne partitionne pas l'ensemble des solutions, puisque l'algorithme ne force pas le début des trajets. Une même solution peut être optimale sur plusieurs branches de l'arbre de recherche.

Nous pouvons remarquer sur l'illustration précédente que le calcul correspondant à l'interdiction de c_2, c_1 n'a pas besoin d'être effectué. En effet, l'algorithme a déjà trouvé une solution au problème interdisant c_1 uniquement, et la solution obtenue S_2 ne comporte pas c_2. Vu que S_2 est optimale dans l'ensemble des trajets ne comportant pas c_1, elle est également optimale dans l'ensemble plus petit (car plus restreint) des trajets ne comportant ni c_1 ni c_2.

Dans le cas général, il se peut qu'une solution optimale dans l'ensemble des solutions s'écrivant $\neg c$ soit également optimale dans l'ensemble $(\neg c, \neg c_1, \dots)$: il suffit qu'elle ne contienne pas les autres courses interdites $(c_1, \dots)$. Vu que le deuxième ensemble

est inclus dans le premier, une solution optimale dans le premier et appartenant au deuxième est nécessairement optimale dans le deuxième.

L'algorithme prend ces cas en compte en vérifiant avant chaque calcul, pour une contrainte $(\neg \mathtt{c}_1, \ldots, \neg \mathtt{c}_n)$ donnée, s'il n'existe pas déjà de solution optimale dans un ensemble moins restreint qui respecte ladite contrainte. Cela implique un parcours de toutes les solutions déjà calculées, avec une vérification de l'inclusion des courses interdites pour les solutions calculées dans les courses que l'on souhaite interdire.

2.5 Variantes de l'algorithme

Nous avons ajouté plusieurs variantes à l'algorithme. La première concerne le type de contraintes imposées à chaque itération. La deuxième concerne la condition d'arrêt. Une dernière variante permet d'augmenter le nombre de trajets obtenus.

2.5.1 Variation des types de contraintes

L'algorithme décrit précédemment se base sur l'interdiction de courses, mais nous avons développé un outil flexible, permettant de gérer d'autres types d'interdictions. Ainsi, il est possible de décliner l'algorithme en représentant un trajet sous l'une des formes suivantes :

- une succession d'itinéraires,
- une succession de lignes,
- une succession de modes,
- une succession d'opérateurs.

L'algorithme est alors reproduit à l'identique en remplaçant les contraintes de type $\neg \mathtt{c}$ en contraintes de type $\neg \mathtt{i}$, $\neg \mathtt{l}$, $\neg \mathtt{m}$ ou $\neg \mathtt{o}$, portant respectivement sur les itinéraires, lignes, modes ou opérateurs.

Avec ces types d'interdictions, l'arbre de recherche n'est alors plus du tout complet et omet une grande partie des solutions réalisables. Cependant, dans le cas de l'interdiction d'itinéraires ou de lignes, les solutions omises restent très similaires aux solutions déjà trouvées puisqu'elles empruntent les mêmes lignes de transport en commun. En revanche, il est évident que l'interdiction de modes ou d'opérateurs contraindra beaucoup le calcul, et le terminera souvent avec peu voire aucune alternatives viables, en particulier dans des zones géographiques où peu d'opérateurs ou de modes sont disponibles.

2.5.2 Condition d'arrêt

L'algorithme est capable de s'arrêter selon l'une de ces trois conditions :

- après avoir effectué un nombre K configurable d'itérations de la boucle WHILE dans l'algorithme 5, c'est-à-dire après avoir trouvé les K « meilleures » solutions,

- après avoir effectué un nombre configurable de calculs d'itinéraire. En effet, à chaque itération, un ou plusieurs calculs sont nécessaires pour obtenir $\mathsf{f}(\mathcal{S})$,
- après un certain temps de calcul.

Dans les trois cas, les trajets retournés sont tous les trajets trouvés pendant l'algorithme, c'est-à-dire l'union $\mathcal{S} \cup \mathcal{S}_a$.

2.5.3 Conservation des labels de fin de calcul

L'algorithme de Dijkstra utilisé dans le calcul d'itinéraire conserve un unique label associé à chaque nœud du graphe : lorsqu'un nouveau label atteint un sommet, il est éliminé s'il est moins bon que le label existant, il élimine ce dernier sinon. Par défaut, cette règle de dominance est également appliquée au nœud de destination finale. Nous proposons de ne pas appliquer cette règle, conservant ainsi tous les labels atteignant le nœud destination. Ces labels n'étant pas propagés, l'impact sur le temps de calcul est nul. Par contre, cette technique permet de générer plusieurs trajets en un seul appel au calculateur.

De plus, la partie de marche à pied finale étant traitée en amont du calcul transport en commun, comme expliqué dans le chapitre I, plusieurs nœuds destinations existent dans notre cas. Nous proposons de retourner l'ensemble de tous les trajets obtenus à tous ces sommets.

2.6 Traitement du critère lexicographique

En option, le calcul graphe *reverse* monolabel, cherchant à trouver le trajet partant au plus tard, peut être fait pour chaque alternative trouvée lors de l'exécution de la méthode monolabel, avec les mêmes interdictions de courses, itinéraires, lignes. Cela double le nombre de calculs d'itinéraire effectués et donc le temps CPU de l'algorithme.

Nous avons également codé une option intermédiaire qui ne fait le calcul graphe *reverse* que pour les k meilleures solutions, et pas pour chaque solution potentielle trouvée.

3 Génération de trajets avec l'algorithme multilabel

Dans cette section, nous allons décrire le principe général de l'algorithme multilabel permettant de générer des solutions diversifiées, puis décrire précisément l'algorithme, accompagné de quelques améliorations et optimisations effectuées pour diminuer les temps d'exécution CPU et obtenir de meilleurs résultats.

3.1 Principe général

Plutôt que de se baser sur une répétition de calculs d'itinéraire, l'objectif de l'algorithme multilabel est de **générer un grand nombre de solutions** assez diverses en un seul calcul. Pour cela, nous avons utilisé et modifié la version multilabel du calculateur d'itinéraire, décrite dans le chapitre IV, section 3.

L'idée principale de l'algorithme est de **conserver une liste de K ou moins labels** par nœud, pour assurer qu'il y ait K solutions sur le nœud final. Si K est élevé, il y aura beaucoup de solutions générées, mais le nombre de labels calculés et donc le temps de calcul risquent d'exploser. Cela pose alors plusieurs questions.

- Comment décider du nombre K ?
- Une fois la limite atteinte, comment décider des labels à conserver sur le nœud lors d'une nouvelle extension ?
- Avant que la limite soit atteinte, faut-il garder tous les labels calculés ?
- Quand arrêter le calcul ?

L'algorithme est l'**algorithme multilabel** présenté dans la section 3 du chapitre IV, auquel est rajouté une partie permettant de limiter le nombre de labels par nœud.

Une fois la limite de labels sur un nœud atteinte, il est naturel de conserver les meilleurs labels en priorité. Nous avons choisi d'utiliser la somme pondérée du calcul monolabel comme indicateur de la qualité d'un label. Nous l'appelons **coût** d'un label. Quand un nouveau label est calculé sur un nœud ayant déjà atteint la limite de K labels, ou bien il a un coût supérieur à tous les autres, et il n'est pas rajouté, ou bien il remplace le label de coût le plus élevé.

3.2 Description de l'algorithme

Nous décrivons ici l'algorithme de diversité multilabel dans sa forme basique, avant une série d'optimisations exposées plus loin dans la présente section.

3.2.1 Ressources

L'algorithme de diversité multilabel utilise les mêmes ressources que l'algorithme monolabel, à savoir un coût c, somme pondérée de plusieurs critères déterminant la qualité du label et un temps t permettant d'utiliser le graphe *time-dependant*. Les autres attributs de label décrits dans le chapitre IV concernant l'algorithme monolabel sont également utilisés.

3.2.2 Règles de dominance, d'extension et d'arrêt

Il n'y a pas de règle de dominance : tous les labels étendus sur un nœud sont conservés indépendamment des labels déjà calculés sur ce nœud. Quels que soient les labels $\mathsf{L}_1, \mathsf{L}_2$ comparés, $\mathsf{L}_1 \prec \mathsf{L}_2$ est faux.

La règle d'extension des labels est identique à l'extension dans le cas de l'algorithme monolabel : c et t se calculent de la même façon, avec les mêmes paramètres.

La condition d'arrêt de l'algorithme multilabel reste la même : l'algorithme s'arrête lorsque le tas binaire est vide.

3.2.3 Limite de nombre de labels par nœud

L'algorithme multilabel utilisé est décrit dans le chapitre IV, algorithme 4. Cependant, la fonction TryAddNewLabel, décrite par l'algorithme 3 dans le cadre de l'algorithme multilabel originel, est modifiée. Elle est remplacée par l'algorithme 6, et permet de ne garder qu'un nombre maximum K de labels par nœud.

Nous notons Labels.GetMaxLabel() la méthode renvoyant le label de coût maximum d'une liste de labels. Son implémentation est de complexité $\mathcal{O}(\mathsf{K})$: l'algorithme parcourt tous les labels du nœud, au maximum K.

La première boucle **for** de l'algorithme 6 est strictement identique à celle de l'algorithme 3, et permet de vérifier si le label testé est dominé par (ou domine) un label existant sur le nœud. En l'occurrence, dans cette version basique de l'algorithme, ce n'est jamais le cas.

Ensuite, dans le cas où il y a strictement moins de K labels sur le nœud, le label est ajouté directement dans la liste des labels du nœud.

Dans le cas contraire, son coût est comparé avec celui du label L_M le plus couteux du nœud : s'il est plus élevé ou égal, il est rejeté. Sinon il est ajouté à la liste, et L_M en est supprimé. De plus, dans ce dernier cas, L_M est marqué comme dominé afin de ne pas être pris en compte lorsqu'il sort du tas binaire.

3.3 Optimisations

Afin de diminuer les temps de calcul, nous avons mis en place plusieurs optimisations permettant de rejeter certaines extensions de labels.

3.3.1 Élimination des labels de coût trop élevé

Nous avons estimé qu'il n'est pas nécessaire de conserver des labels dont les coûts sont trop élevés. Par exemple, si deux trajets arrivent au même arrêt, et si l'un des deux arrive avec un coût supérieur de plus de 35 % au coût de l'autre, il est peu probable qu'on souhaite le garder : quand bien même il apporterait une alternative intéressante, le trajet est trop mauvais. Nous avons donc mis en place une règle de dominance permettant à des labels trop couteux d'être éliminé avant que la limite de K soit atteinte.

Pour cela nous introduisons un paramètre (nombre réel) $\gamma_2 > 1$. Soit $\mathsf{L}_1(\mathsf{c}_1)$ et $\mathsf{L}_2(\mathsf{c}_2)$ deux labels à comparer. Le label L_2 est dominé par L_1 (et donc éliminé) si son coût est plus élevé que celui de L_1 multiplié par γ_2.

Algorithm 6 TryAddNewLabel($\mathfrak{n}, \mathsf{L}$)

```
for all L_n ∈ n.Labels do
  if L_n ≺ L then
    return false
  else if L ≺ L_n then
    L_n.isDominated ← true
    n.Labels.Remove(L_n)
  end if
end for
addLabel ← true
if n.Labels.size() = K then
  L_M ← Labels.GetMaxLabel()
  if L_M.Cost > L.Cost then
    n.Remove(L_M)
    L_M.isDominated ← true
  else
    addLabel ← false
  end if
end if
if addLabel then
  n.Labels.Add(L)
  allLabels.Add(L)
end if
return true
```

$$L_1 \prec L_2 \Leftrightarrow \gamma_2 \cdot c_1 < c_2$$

Nous avons choisi un coefficient multiplicatif car une alternative de trente minutes à un trajet de dix minutes n'est pas acceptable, mais une alternative de 3h20 à un trajet de 3h est raisonnable.

C'est une heuristique dont l'objectif est de diminuer les temps de calcul, mais qui risque de détériorer les résultats, car rien ne dit qu'un label plusieurs minutes plus lent au début ne peut pas se « rattraper » à la fin et donner naissance à un trajet alternatif intéressant. Comme le coût c d'un label en début de calcul est relativement peu élevé, la marge $(\gamma_2 - 1) \cdot c$ est d'autant plus faible. Pour diminuer les risques d'éliminer un label intéressant, nous introduisons également un coefficient additif γ_1, qui représente une marge en dessous de laquelle nous n'appliquerons pas cette règle de dominance.

Nous avons donc défini une règle de dominance entre les labels qui utilise un coefficient additif γ_1 et un coefficient multiplicatif γ_2.

$$L_1 \prec L_2 \Leftrightarrow \max(c_1 + \gamma_1, \gamma_2 \cdot c_1) < c_2$$

3.3.2 Labels similaires

Il est important de noter qu'il est inefficace de conserver des labels très similaires au même nœud, car leurs extensions seront également similaires. Puisque le calcul de la distance (voir 4) n'est pas instantané, et que le nombre de comparaisons de labels est extrêmement élevé, nous avons choisi de nous baser sur une **fonction de hachage** de la séquence des lignes empruntées pour juger de la similarité entre les labels. Si deux labels comparés au même nœud partagent effectivement la même séquence de lignes, alors l'un des deux sera éliminé au profit du label le moins couteux.

La valeur de hachage h devient une ressource supplémentaire du label, telle que deux labels partageant la même valeur de hachage partagent nécessairement la même séquence de lignes empruntées. Elle est calculée avec une fonction de hachage de type shift-add-xor, décrit dans [Rama1997]. Ce calcul a l'avantage de pouvoir se déduire directement à partir de la valeur de hachage précédente. Les symboles $\oplus, \mathcal{L}_b(), \mathcal{R}_b()$ représente respectivement l'opérateur binaire XOR, le décalage binaire vers la gauche de b bits et le décalage binaire vers la droite de b bits. h est un hachage de la liste des index de lignes, entier identifiant la ligne, compris entre 1 et le nombres de lignes.

La formule suivante permet de calculer la nouvelle valeur de hachage, qui est mise à jour lors de **l'extension** d'un label sur un arc GETIN uniquement, vers un nœud ITI dont la ligne est identifiée par un entier i.

$$h' = h \oplus (\mathcal{L}_5(h) + \mathcal{R}_2(h) + i)$$

La règle de dominance pour des labels $L_1(c_1, h_1)$ et $L_2(c_2, h_2)$ partageant la même valeur de hachage devient :

Si $h_1 = h_2$, $L_1 \prec L_2 \Leftrightarrow c_1 < c_2$, sinon appliquer la dominance décrite précédemment utilisant γ_1 et γ_2.

3.3.3 Répétition d'arrêts

Étant donné que les labels sous-optimaux sont conservés, il est possible qu'un label s'étende vers un nœud MARCHE qu'il a déjà parcouru une fois. Or une alternative à un trajet qui n'en diffère que par l'ajout d'une boucle à pied ou en transport en commun n'est pas acceptable.

Pour réduire encore les calculs inutiles, nous avons donc décidé d'implémenter un filtre à l'extension des labels.

Lorsqu'un label est étendu vers un nœud MARCHE, s'il fait partie des nœuds MARCHE précédemment parcourus par le label, l'extension est interdite.

Nous introduisons pour cela une ressource L_s listant les derniers nœuds MARCHE parcourus. Lors de l'extension d'un arc GETOFF vers un nœud MARCHE n, l'extension est interdite si $n \in L_s$, sinon n est ajouté à L_s.

3.3.4 Répétition de lignes

Une dernière heuristique d'optimisation consiste à éviter qu'une ligne soit répétée dans le même label.

En effet, il arrive très fréquemment que deux labels aient deux valeurs de hachage différentes et des coûts proches, sans que le deuxième soit intéressant. Par exemple, quand à partir du premier, le deuxième descend et remonte sur la prochaine course de la ligne qu'il vient de quitter. Pour éviter ce cas de figure, nous ajoutons un autre filtre à l'extension des labels.

Lorsqu'un label est étendu vers un nœud ITI, si la ligne associée a déjà été empruntée par le label, l'extension est interdite.

Nous introduisons pour cela une ressource L_l listant les dernières lignes empruntées. Lors de l'extension d'un arc GETIN vers un nœud ITI dont l'itinéraire est sur la ligne l, l'extension est interdite si $l \in L_l$, sinon l est ajouté à L_l.

Une contrepartie de cette heuristique concerne les trajets optimaux qui contiennent plusieurs étapes de la même ligne, mais sur des itinéraires différents. Par exemple, il est parfois nécessaire de descendre d'un RER et remonter sur la même ligne, pour atteindre des arrêts qui ne seraient autrement pas desservis. Ces trajets ne seront malheureusement pas renvoyés par l'algorithme. Néanmoins, cette heuristique est indispensable pour éviter une surabondance de labels inutiles.

3.3.5 Arrêt du calcul et complexité

Dès qu'un label final est calculé, celui-ci sert de borne de dominance pour toutes les nouvelles extensions. Cette borne est remplacée par le label final de coût minimum, afin d'accélérer l'arrêt du calcul. Tout label dominé par le meilleur label final est éliminé. L'algorithme se termine car chaque extension ajoute une valeur entière non nulle au coût du label.

Théoriquement, l'algorithme est NP-complet. Cependant, le fait de limiter le nombre de labels par nœud et les divers garde-fous implémentés permettent d'obtenir des temps de calcul raisonnables, comme le montrent les expérimentations.

3.4 Traitement du critère lexicographique

Nous avons fait le choix de ne pas faire de calcul graphe *reverse* après le calcul direct en multilabel. Nous supposons que la diversité des labels générés par cet algorithme, accompagnée du décalage des horaires au plus tard (voir VI, section 2.4), permet d'obtenir la majorité des solutions optimisant le critère **lexicographique**. De plus, réaliser des calculs multilabels supplémentaires augmenterait fortement des temps de calculs déjà assez grands.

Cependant une étude plus poussée, que nous n'avons pas effectuée par manque de temps, pourrait établir quel horaire d'arrivée utiliser pour un calcul graphe *reverse* et comment l'optimiser de manière à obtenir un ensemble de solutions plus grand et plus pertinent relativement au critère de **départ au plus tard**, en des temps raisonnables.

4 Méthode de sélection de trajets diversifiés

Dans cette section, nous proposons des méthodes permettant de sélectionner un nombre restreint, en pratique entre trois et cinq, de trajets parmi un grand nombre de solutions préalablement obtenues. Il ne s'agit pas simplement de prendre les trois meilleurs trajets selon certains critères, mais d'utiliser leur diversité, c'est-à-dire ce qui les différencie deux à deux, pour en obtenir un panel représentatif. Il ne s'agit pas non plus de sélectionner les solutions en se basant uniquement sur des mesures de diversité. En effet, si une solution est largement moins rapide que la solution optimale, il n'est pas pertinent de la montrer à l'utilisateur, car il est très probable qu'il la rejettera au final. Ainsi nous désirons par exemple que si les trois solutions les plus rapides utilisent toutes trois le bus de l'opérateur X, mais qu'il existe une solution presque aussi bonne qui utilise le train de l'opérateur Y, cette dernière soit présente parmi les trajets sélectionnés, car elle satisfait à la fois des critères de qualité et de contribution à la diversité.

Avant de présenter les algorithmes, il nous faut introduire une mesure quantitative de la diversité. Nous proposons pour cela de définir une distance, mesurant la différence entre deux solutions.

4.1 Définition d'une distance entre trajets

Nous cherchons à déterminer, pour deux trajets S_1, S_2, une fonction de ces trajets qui renverrait un nombre $d(S_1, S_2) \in [0, 1]$, et qui doit mesurer la non-similarité de ces trajets entre eux. Ainsi, lorsque nous définissons cette distance, nous tentons de quantifier ce qui fait qu'un trajet est plus ou moins différent d'un autre. Cette distance ne s'applique qu'à des trajets partageant les mêmes points de départ et d'arrivée.

Rappelons qu'un trajet en transport en commun se décompose en étapes de transport en commun, qui ont un certain nombre d'attributs, tels que la course, l'itinéraire, la ligne, le mode et l'opérateur. Des étapes communes doivent diminuer la distance. Si deux trajets ont toutes leurs étapes communes, la distance doit être nulle. Si aucun attribut d'aucune étape n'est identique, la distance doit être maximale (égale à 1).

La différence entre deux trajets se définit en fonction des attributs d'étapes : nous pouvons parler de différence en termes de courses, d'itinéraires, de lignes, de modes et d'opérateurs. L'ordre des étapes est important, un trajet qui commence par 30 minutes de bus et termine par 30 minutes de métro sera considéré comme totalement différent, selon les modes, d'un trajet qui commence par 30 minutes de métro et se termine par 30 minutes de bus. Le décalage temporel entre deux trajets importe peu : deux trajets empruntant les mêmes itinéraires successifs dans les mêmes proportions seront considérés comme similaires en termes d'itinéraires, même s'ils ne sont pas aux mêmes horaires de la journée. Par exemple, les trois trajets illustrés en figure 30 doivent avoir une distance itinéraire nulle.

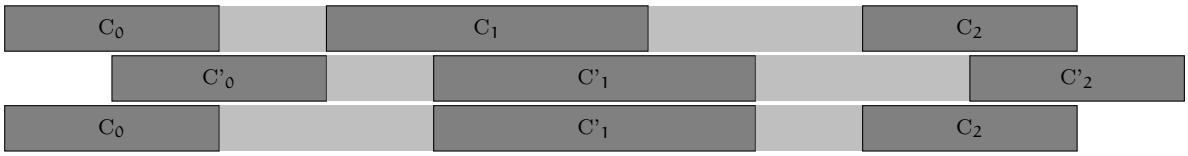


FIGURE 30 – trois trajets aux mêmes itinéraires mais aux courses variant légèrement

L'importance relative de chaque type d'attribut d'étape doit pouvoir être configurable. Par exemple il faut pouvoir choisir de privilégier une différence de lignes ou une différence de modes plutôt que des différences d'itinéraires.

Définition Pour répondre à ces préceptes, nous avons choisi de définir la distance comme une somme pondérée des six fonctions suivantes, que nous appellerons abusivement des « distances ». En effet une distance sur un ensemble E est une fonction de E^2 vers $\mathbb{R}^+$, symétrique, qui vérifie l'inégalité triangulaire, et la séparation : $f(x, y) = 0 \Leftrightarrow x = y$. Les distances que nous définissons ici ne respectent pas la séparation, sauf si l'on considère comme ensemble de départ non pas les trajets, mais leur « représentation proportionnelle » (voir définition ci-dessous).

- Distance course d_c qui mesure l'écart des solutions en termes de courses empruntées.
- Distance itinéraire d_i qui mesure l'écart des solutions en termes d'itinéraires empruntés.

- Distance ligne d_l qui mesure l'écart des solutions en termes de lignes empruntées.
- Distance mode d_m qui mesure l'écart des solutions en termes de modes empruntés.
- Distance opérateur d_o qui mesure l'écart des solutions en termes d'opérateurs empruntés.
- Distance mode/opérateur d_{mo} qui mesure l'écart des solutions en termes des couples mode/opérateurs empruntés.

Chacune de ces fonctions que l'on notera d_a pour un attribut quelconque a (course, itinéraire, ligne mode opérateur ou mode/opérateur) prend en argument un couple de trajets réalisables dans le réseau considéré, et a comme valeur un réel entre 0 et 1.

Définition 35 *Soit* T *un trajet de transport en commun composé d'étapes en transport en commun* $(e_i)_{i\in[1,n]}$. *La* ***durée proportionnelle*** δ_i *de l'étape* e_i *de* T *est la proportion de sa durée sur la somme des durées des étapes de* T.

Précisons que les étapes d'attente ou de marche à pied ne sont pas considérées dans le calcul des durées proportionnelles.

Définition 36 *Soit* T *un trajet de transport en commun composé d'étapes en transport en commun* $(e_i)_{i\in[1,n]}$. *La* ***date de départ proportionnelle*** $\tau_i \in [0,1]$ *de l'étape* e_i *de* T *est la somme des durées proportionnelles des étapes* $e_1, \ldots, e_{i-1}$.

$$\tau_i = \sum_{j=1}^{i-1} \delta_j$$

Illustration La figure 31 présente un exemple de trajet. Le trajet dure 1h25, et est composé de trois étapes de durée totale 1h10. Selon les définitions précédentes, nous avons donc :

$$\delta_1 = 2/7\ ,\ \delta_2 = 1/2\ ,\ \delta_3 = 3/14$$
$$\tau_1 = 0\ ,\ \tau_2 = 2/7\ ,\ \tau_3 = 11/14$$

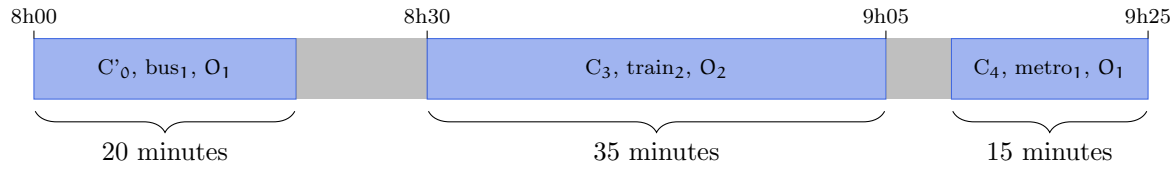


FIGURE 31 – Exemple d'un trajet en trois étapes de durée 1h25

Définition 37 *Un attribut* a *d'étape est une fonction qui a toute étape de trajet associe une valeur dans un ensemble de valeurs de cet attribut.*

Les attributs d'étape peuvent être la course à valeur dans l'ensemble des courses, l'itinéraire à valeur dans l'ensemble des itinéraires, et de même pour la ligne, le mode, l'opérateur, ou le mode/opérateur.

Définition 38 *Par extension, pour un attribut* a *et un trajet* T*, la fonction attribut* a_T *est une fonction de* $[0,1]$ *dans l'ensemble des valeurs possibles de l'attribut (image de* a*), telle que* $a_T(\tau)$ *est égal à la valeur de la fonction attribut pour l'étape en cours, au sens des dates de départ proportionnelles, à la date proportionnelle* τ*.*

Notons que cette définition ne définit pas clairement la fonction aux points qui délimitent deux étapes du trajet. Par convention, mais sans incidence sur les calculs à suivre, nous fixons la valeur de la fonction égale à l'attribut de l'étape d'après si elle existe.

Par exemple, pour le trajet T de la figure 31, l'attribut « course » c_T est défini par :

- $c_T(\tau) = C'_0$ pour $\tau \in [0, 2/7[$,
- $c_T(\tau) = C_1$ pour $\tau \in [2/7, 11/14[$,
- $c_T(\tau) = C_4$ pour $\tau \in [11/14, 1]$.

Définition 39 *Pour un attribut quelconque* a*, la fonction distance correspondante* d_a *est définie de la manière suivante. Soient deux trajets* T_1, T_2*. Nous considérons l'union des dates de départ proportionnelles des deux trajets et nous notons* $(\tau_i)_{i\in[1,n]}$ *la suite des ces dates, triée par ordre croissant. Pour tout* $i \in [1, n]$*, la période entre* τ_i *et* τ_{i+1} *correspond à une seule étape* $e_{i,1}$ *du trajet* T_1 *et une seule étape* $e_{i,2}$ *du trajet* T_2*. Nous notons alors* $\chi_{T_1,T_2}(i)$ *une fonction de valeur 1 si* $a(e_{i,1}) \neq a(e_{i,2})$ *et 0 sinon. Alors,*

$$d_a(T_1, T_2) = \sum_{i=1}^{n-1} \chi_{T_1,T_2}(i) \cdot (\tau_{i+1} - \tau_i)$$

Remarquons que la fonction d_a peut, de manière équivalente, être définie par une intégrale. Nous introduisons la fonction $X_{T_1,T_2} : \tau \in [0,1] \to \{0,1\}$ telle que
$X_{T_1,T_2}(\tau) = 1$ si $a_{T_1}(\tau) \neq a_{T_2}(\tau)$,
$X_{T_1,T_2}(\tau) = 0$ sinon. Alors :

$$d_a(T_1, T_2) = \int_0^1 X_{T_1,T_2}(\tau) d\tau$$

Définition 40 *Nous définissons sur l'ensemble des trajets réalisables une distance* d *par :*

$$d(T_1, T_2) = \sum_{a\in\{c,i,l,o,m,mo\}} \lambda_a \cdot d_a(T_1, T_2)$$

où les λ_a *sont des coefficients positifs non nuls et de somme égale à un.*

Illustration La figure 32 décrit deux trajets résultant d'une requête. Le premier correspond à l'exemple précédent. Le deuxième est une succession de 20 minutes de bus (même itinéraire mais pas même course que le premier bus du premier trajet), 20 minutes de train (ligne différente de celle du premier trajet) et 30 minutes de bus. La figure 33 dépeint la représentation proportionnelle de ces deux trajets. Il est facile d'en déduire les valeurs des distances suivantes :

- Distance course : $d_c = 1$
- Distance itinéraire : $d_i = {}^{10}/_{14}$
- Distance ligne : $d_l = {}^{10}/_{14}$
- Distance mode : $d_m = {}^{6}/_{14}$
- Distance opérateur : $d_o = {}^{3}/_{14}$
- Distance mode/opérateur : $d_{mo} = {}^{6}/_{14}$

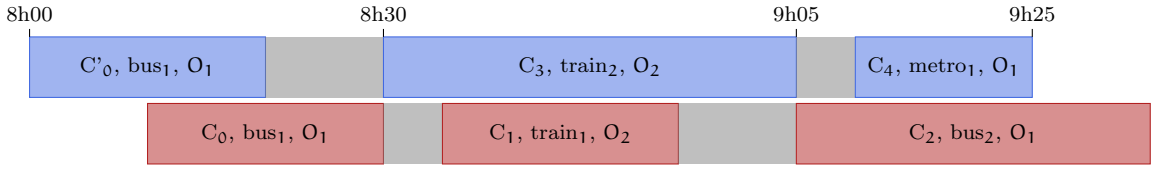


FIGURE 32 – Exemple de deux trajets résultant d'une requête

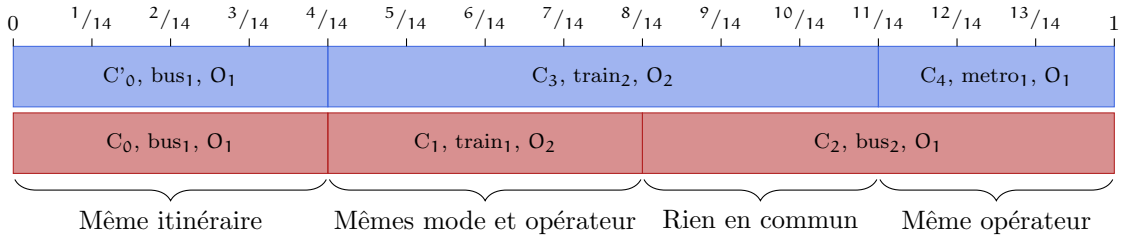


FIGURE 33 – Illustration du calcul de la distance entre les deux trajets de la figure 32

4.1.1 Propriétés de la fonction d

Nous démontrons ici que d est une fonction symétrique et qui respecte l'inégalité triangulaire, satisfaisant donc deux des conditions requises pour être qualifiée de distance. En revanche, comme déjà mentionné, d ne respecte pas la séparation, car deux trajets ayant la même succession de courses avec les même temps proportionnels pourraient être différents, dans des cas rares.

Proposition 1 *La fonction* d *est une fonction symétrique.*

$$d(T_1, T_2) = d(T_2, T_1)$$

PREUVE

Par définition :

$$d(T_1, T_2) = \sum_{a \in \{c,i,l,o,m,mo\}} \lambda_a \cdot d_a(T_1, T_2)$$

Il suffit donc de montrer la symétrie de chaque fonction d_a. Utilisons pour cela la définition sous forme d'intégrale :

$$d_a(T_1, T_2) = \int_0^1 X_{T_1,T_2}(\tau) d\tau$$

Or $X_{T_1,T_2} = X_{T_2,T_1}$ car pour tout $\tau \in [0, 1]$, ou bien $X_{T_1,T_2}(\tau) = 1$, donc T_1 et T_2 n'ont pas la même valeur d'attribut au temps τ, et donc $X_{T_2,T_1}(\tau) = 1$, ou bien $X_{T_1,T_2}(\tau) = 0$ donc T_1 et T_2 ont la même valeur d'attribut au temps τ et $X_{T_2,T_1}(\tau) = 0$. □

Proposition 2 *La fonction* d *respecte l'inégalité triangulaire.*

$$d(T_1, T_3) \leq d(T_1, T_2) + d(T_2, T_3)$$

Preuve

De nouveau, il suffit de le montrer pour une distance d'attribut d_a. Soient trois trajets comparables T_1, T_2, T_3. Définissons une suite finie de temps proportionnels $(\tau_i)_{i \in [1,n]}$ comme l'union des dates de départ proportionnelles des trois trajets. Alors pour tout $i \in [1, n-1]$, la période proportionnelle $[\tau_i, \tau_{i+1}]$ correspond à une seule étape de chaque trajet : $e_{1,i}$ pour T_1, $e_{2,i}$ pour T_2 et $e_{3,i}$ pour T_3. Montrons que :

$$\chi_{T_1,T_3}(i) \leq \chi_{T_1,T_2}(i) + \chi_{T_2,T_3}(i)$$

Vu que les χ n'ont pour valeur que zéro ou un, le seul cas où l'inégalité pourrait ne pas être vérifiée est le cas $\chi_{T_1,T_3}(i) = 1$ et $\chi_{T_1,T_2}(i) = \chi_{T_2,T_3}(i) = 0$. Ce cas ne peut pas se produire car il impliquerait à la fois $a(e_{1,i}) \neq a(e_{3,i})$ et $a(e_{1,i}) = a(e_{2,i}) = a(e_{3,i})$. Donc :

$$\begin{aligned}
d_a(T_1, T_3) &= \sum_{i=1}^{n-1} \chi_{T_1,T_3}(i) \cdot (\tau_{i+1} - \tau_i) \\
&\leq \sum_{i=1}^{n-1} (\chi_{T_1,T_2}(i) + \chi_{T_2,T_3}(i)) \cdot (\tau_{i+1} - \tau_i) \\
&\leq \sum_{i=1}^{n-1} \chi_{T_1,T_2}(i) \cdot (\tau_{i+1} - \tau_i) + \sum_{i=1}^{n-1} \chi_{T_2,T_3}(i) \cdot (\tau_{i+1} - \tau_i) \\
&\leq d_a(T_1, T_2) + d_a(T_2, T_3)
\end{aligned}$$

□

4.2 Sélection de solutions

Les deux méthodes de sélection développées se basent toutes les deux sur la distance d définie précédemment et sur des algorithmes de regroupement de données (*clustering*). Ainsi, nous nous efforçons de classifier les solutions obtenues en groupes de solutions similaires, au sens de la distance d. Une fois cette classification obtenue, notre ensemble final de solutions est l'ensemble des meilleures solutions de chaque groupe. Le nombre de groupes peut être fixe, ou dépendre du nombre et de la nature des solutions à classifier.

Notations Pour décrire les algorithmes, nous supposons disposer d'une liste de N trajets $\{\mathsf{T}_1, \ldots, \mathsf{T}_\mathsf{N}\}$, ordonnée du meilleur au moins bon selon un ordre défini par une somme pondérée de critères. Pour alléger les notations, nous noterons $\mathsf{d}_{\mathsf{i},\mathsf{j}} = \mathsf{d}(\mathsf{T}_\mathsf{i}, \mathsf{T}_\mathsf{j})$ la distance entre deux trajets. Le résultat d'un algorithme de sélection est un sous-ensemble de l'ensemble initial.

4.2.1 K-Means

Le premier algorithme de clustering implémenté est la méthode des k-moyennes, dont la version classique est attribuée à [Macq1967], affinée par [Hart1975]. Cet algorithme nécessite de connaître k, le nombre de groupes que l'on souhaite obtenir. Schématiquement, il consiste en une initialisation des centroïdes de chaque groupe, aléatoire par exemple, ou plus intelligente, puis à répéter jusqu'à convergence une étape de réaffectation des éléments à chaque groupe, en fonction de leur distance au centroïde, et une étape de recalcul du centroïde.

Dans notre cas, la notion de centroïde d'un groupe de solutions n'a aucun sens. En revanche, en remarquant que le centroïde ne sert qu'à faire un calcul de distance, nous pouvons aisément remplacer la distance au centroïde par la moyenne des distances à chaque objet de la partition. Hormis cette mise à jour, l'algorithme implémenté reproduit le schéma classique de la méthode des k-moyennes.

Description de l'algorithme Notons $\mathsf{c}_1, \ldots, \mathsf{c}_\mathsf{k}$ les clusters finaux, résultant de l'algorithme. Nous supposons l'existence des méthodes suivantes.

- *cluster*.Add(*trip*) : ajoute *trip* dans un *cluster*.
- distance(*trip*, *cluster*) : calcul de la distance d'un trajet *trip* à un *cluster* par la moyenne des distances de ce trajet aux trajets du *cluster*. Complexité proportionnelle au nombre de trajets dans le *cluster*, fois la complexité du calcul de distance d'un trajet à un autre.
- findFurthestTrip(*trips*, *clusters*) : trouve et renvoie le trajet de *trips* le plus éloigné de tous les *clusters*, c'est-à-dire maximisant la somme des distances à chaque cluster. Complexité en taille de *trips* x nombre total de trips des *clusters*.
- *trip*.nearestCluster() : trouve, parmi tous les clusters existant, le cluster le plus proche de *trip*. Complexité en nombre de trajets dans tous les clusters.
- *trip*.currentCluster() : renvoie dans quel cluster est affecté *trip*.

L'algorithme 7 décrit l'initialisation de notre implémentation des K-Means. D'abord le trajet T_1 est placé dans le cluster c_1. Puis chaque cluster est initialisé avec le trajet le plus éloigné des clusters déjà initialisés. Ensuite, chaque trajet non affecté à un cluster est rajouté au cluster le plus proche. Lorsque l'initialisation est terminée, chaque trajet est affecté à un cluster, de façon à maximiser les distances entre clusters.

Algorithm 7 Algorithme des k-moyennes : initialisation des clusters

```
c_1, ..., c_k ← ∅, ..., ∅
c_1.Add(T_1)
tripsToInit ← (T_2, ..., T_N)
for j ← 2; j ≤ k; j ← j + 1 do
  T ← findFurthestTrip(tripsToInit, (c_1, ..., c_{j-1}))
  c_j.Add(T)
  tripsToInit.remove(T)
end for
while tripsToInit.size() > 0 do
  T ← tripsToInit.pop_back()
  c ← T.nearestCluster()
  c.Add(T)
end while
```

L'étape de réaffectation est décrite par l'algorithme 8. Une boucle « tant que », répétée jusqu'à stabilisation des clusters, trouve le cluster le plus proche de chaque trajet. Dans une deuxième boucle, les réaffectations correspondantes sont réalisées.

Enfin, l'algorithme renvoie les meilleures solutions de chaque cluster.

Algorithm 8 Algorithme des k-moyennes : réaffectation

```
changed ← 1
while changed = 1 do
  tripsToReaffect ← ∅
  changed ← 0
  for all T ∈ (T_1, ..., T_N) do
    c_n ← T.nearestCluster()
    c_c ← T.currentCluster()
    if c_c ≠ c_n then
      tripsToReaffect.Add(T, c_n, c_c)
      changed ← 1
    end if
  end for
  for all (T, c_n, c_n) ∈ tripsToReaffect do
    c_c.Remove(T)
    c_n.Add(T)
  end for
end while
return c_1.best(), ..., c_k.best()
```

4.2.2 Élimination ascendante hiérarchique

Un autre algorithme classique de clustering est le regroupement hiérarchique, ou classification ascendante hiérarchique, illustré par exemple dans [Sibs1973]. Étant donné un ensemble E de n éléments, le but de l'algorithme est de répartir ces éléments dans un certain nombre (inférieur à n) de classes. L'algorithme procède par la création d'une hiérarchie de classes, partant d'autant de classes qu'il y a d'éléments dans E, chaque classe étant réduite à un élément. A chaque étape les deux classes les plus proches l'une de l'autre sont regroupées, ces regroupements successifs constituant le résultat de l'algorithme, souvent représenté graphiquement par un dendogramme. La façon de rapprocher les classes peut se faire soit au sens de la distance minimale, c'est l'algorithme de single-link clustering, introduit dans [Sibs1973], soit au sens de la distance maximale, c'est l'algorithme de complete-link clustering, introduit dans [Defa1977], ou encore d'autres manières.

Dans notre cas, nous avons défini une distance pour les trajets deux à deux, mais comment définir la distance entre deux ensembles de trajets ? Nous pourrions choisir la distance moyenne, maximale ou minimale, mais étant donné que nous construisons ces regroupements de trajets dans le but de n'en afficher qu'un seul par regroupement, à savoir le meilleur en terme de somme pondérée, il nous a paru pertinent d'utiliser comme distance d'un ensemble à un autre la distance entre la meilleure solution du premier et la meilleure solution du deuxième. Et étant donné que nous ne comptons afficher qu'une seule solution par regroupement, et que les solutions dans un regroupement qui ne sont pas la meilleure n'ont pas d'influence sur la suite du regroupement, nous pouvons les éliminer.

Nous obtenons ainsi un algorithme que nous appelons algorithme d'élimination hiérarchique. Cet algorithme permet d'éliminer en priorité les solutions moins bonnes qui ressemblent à d'autres solutions. Nous en détaillons le fonctionnement dans l'algorithme 9.

Algorithm 9 Algorithme d'élimination ascendante hiérarchique

```
distanceList.Sort()
eliminationList ← ∅
while distanceList ≠ ∅ do
    d_{i,j} ← distanceList.pop_first()
    eliminationList.Add(T_j, T_i, d_{i,j})
    for all d_{k,l} ∈ distanceList do
        if (k = j) ∨ (l = j) then
            distanceList.remove(d_{k,l})
        end if
    end for
end while
return eliminationList
```

Pour cet algorithme, nous appelons distanceList la liste des valeurs de distance $d_{i,j}$. Le résultat de l'algorithme est une liste ordonnant l'élimination de tous les trajets, sauf le meilleur, c'est-à-dire T_1. Cette liste contient des triplets avec comme premier élément le trajet éliminé T_j, comme second élément le trajet duquel il a été rapproché

T_i, nécessairement meilleur que T_j, et comme dernier élément la distance qui les sépare $d_{i,j}$.

4.2.3 Illustration

Considérons un ensemble de cinq trajets $(T_1, T_2, T_3, T_4, T_5)$, ordonnés du meilleur au moins bon. Supposons que le tableau 7 représente les distances entre ces trajets (elles ne sont pas dans $[0, 1]$, mais cela ne change en rien l'algorithme illustré).

	T_1	T_2	T_3	T_4	T_5
T_1	0	28	29	15	77
T_2		0	12	25	50
T_3			0	37	55
T_4				0	62
T_5					0

Tableau 7 – Table de distances

Nous allons voir comment les algorithmes de KMeans et d'EAH se comportent pour cet exemple.

Illustration du K-Means Nous choisissons $k = 3$ pour illustrer cet exemple.

L'étape d'initialisation est constituée des étapes d'affectations des cinq trajets aux trois clusters.

- T_1 est affecté à c_1.
- Le trajet le plus éloigné des clusters non vides, c'est-à-dire de T_1 est T_5, avec comme distance $d_{1,5} = 77$. Il est donc affecté à c_2.
- Le trajet le plus éloigné des clusters non vides, c'est-à-dire de T_1 et T_5, est T_3, il est donc affecté à c_3. En effet, $d_{1,2} + d_{5,2} = 78$, $d_{1,3} + d_{5,3} = 84$, $d_{1,4} + d_{5,4} = 77$.
- Les clusters étant remplis, nous cherchons à affecter T_2 et T_4, qui ne sont pas encore dans un cluster. Les distance minimums pour T_2 et T_4 sont respectivement $d_{2,3} = 12$ et $d_{4,1} = 15$

Ces quatre étapes et la composition des clusters pendant l'initialisation sont résumées dans le tableau 4.2.3.

Étapes :	1	2	3	4
c_1	T_1	T_1	T_1	T_1, T_4
c_2	$\emptyset$	T_5	T_5	T_5
c_3	$\emptyset$	$\emptyset$	T_3	T_3, T_2

Tableau 8 – Illustration des K-Means : initialisation des clusters

A l'étape de réaffectation, une boucle est effectuée sur chaque trajet T_1, T_2, T_3, T_4, T_5, en calculant leur distance à chaque cluster. Les résultats, indiqués dans le tableau 9 montrent qu'il n'y a pas de réaffectation.

T	$d(T, c_1)$	$d(T, c_2)$	$d(T, c_3)$	Réaffectation
$T_1 \in c_1$	7,5	77	28,5	non
$T_2 \in c_3$	26,5	50	6	non
$T_3 \in c_3$	33	55	6	non
$T_4 \in c_1$	7,5	62	31	non
$T_5 \in c_2$	69,5	0	52,5	non

Tableau 9 – Illustration des K-Means : réaffectation

Les solutions sont alors T_1, T_2, T_5, les meilleures de chaque cluster.

Illustration de l'EAH Les étapes successives de l'algorithme d'élimination ascendante hiérarchique sont :

1	distanceList	: ($\boxed{d_{2,3}}$, $d_{1,4}$, $d_{2,4}$, $d_{1,2}$, $d_{1,3}$, $d_{3,4}$, $d_{2,5}$, $d_{3,5}$, $d_{4,5}$, $d_{1,5}$)
	eliminationList	: $\emptyset$
2	distanceList	: ($\boxed{d_{1,4}}$, $d_{2,4}$, $d_{1,2}$, $d_{2,5}$, $d_{4,5}$, $d_{1,5}$)
	eliminationList	: $((T_3, T_2, 12))$
3	distanceList	: ($\boxed{d_{1,2}}$, $d_{2,5}$, $d_{1,5}$)
	eliminationList	: $((T_3, T_2, 12), (T_4, T_1, 15))$
4	distanceList	: ($\boxed{d_{1,5}}$)
	eliminationList	: $((T_3, T_2, 12), (T_4, T_1, 15), (T_2, T_1, 28))$
5	distanceList	: $\emptyset$
	eliminationList	: $((T_3, T_2, 12), (T_4, T_1, 15), (T_2, T_1, 28), (T_5, T_1, 77))$

L'ordre d'élimination obtenu est donc T_3, T_4, T_2, T_5, ce qui est représenté par le dendrogramme de la figure 34. S'il faut retourner 3 trajets à l'utilisateur, les trajets sélectionnés seront T_1, T_5 et T_2.

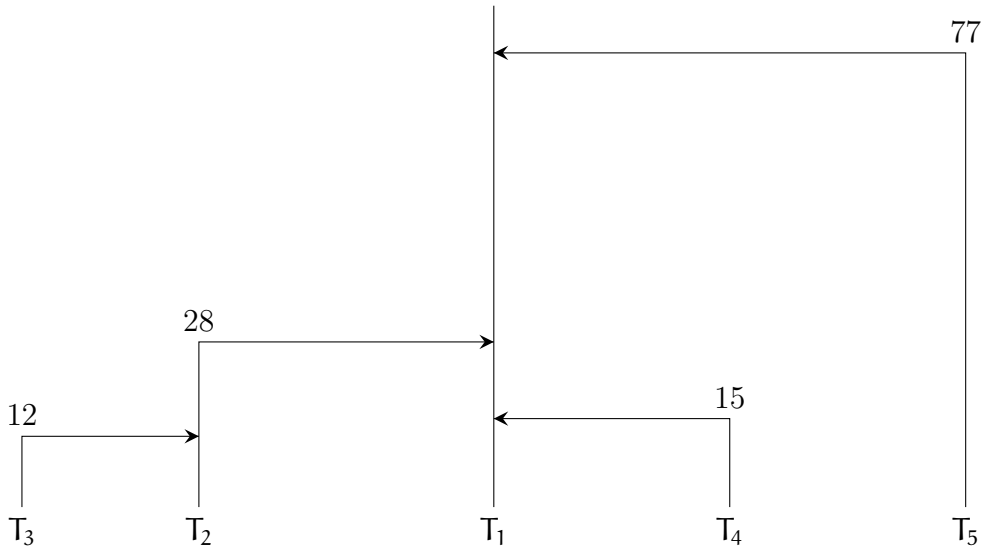


FIGURE 34 – Exemple de l'élimination hiérarchique : dendrogramme

5 Expérimentations numériques

Cette section est consacrée aux résultats d'expérimentations des méthodes de génération et sélection de solutions diversifiées réalisées sur le réseau R1.

Avant ces résultats, nous présentons également les indicateurs de comparaison utilisés pour juger des résultats d'expérimentation et le protocole de test.

5.1 Indicateurs de comparaison

Afin de tester les algorithmes décrits dans les sections précédentes, nous avons établi des indicateurs à la fois de qualité des solutions obtenues, mais aussi de la diversité apportée par l'ensemble de solutions.

5.1.1 Indicateurs de qualité

Pour comparer la qualité de deux ensembles de trajets A et B, nous nous basons sur la somme pondérée temps et nombre de changements qui déterminent les coûts des arcs en monolabel. Nous notons $c(S)$ cette somme pondérée pour une solution S et $c_0(A,B)$ le coût minimal de la meilleure solution des deux ensembles. Nous définissons alors les indicateurs suivants.

- $i_{moy}(A,B)$ comme la moyenne des écarts relatifs des coûts des solutions de A par rapport à $c_0(A,B)$.

$$i_{moy}(A,B) = \frac{1}{|A|}.\sum_{S\in A}\frac{c(S)-c_0(A,B)}{c_0(A,B)}$$

- $i_{max}(A,B)$ comme l'écart relatif de la pire solution de A par rapport à $c_0(A,B)$.

$$i_{max}(A, B) = \max_{S \in A} \left(\frac{c(S) - c_0(A, B)}{c_0(A, B)} \right)$$

5.1.2 Indicateurs de distance

Il ne suffit pas de savoir quel ensemble présente les meilleures solutions sur le critère de somme pondérée. Nous cherchons également à mesurer la diversité des ensembles de solutions.

Pour cela, nous reprenons la définition de distance entre les solutions introduite dans les sections précédentes. Nous définissons l'**apport de diversité** d'une solution a à un ensemble de solutions A comme la distance minimum entre a et les éléments de $A \setminus \{a\}$. Si cette mesure est proche de 0, il existe dans A une solution différente de a, mais qui s'en rapproche. À l'opposé, si cette mesure est proche de un, aucune solution de A ne ressemble à a, et la rajouter apporte beaucoup de diversité.

Le premier indicateur utilisé, noté $\delta(A)$, décrit la diversité d'un ensemble de solutions A. Il est égal à la moyenne de l'**apport de diversité** de chaque solution de A à lui-même. Un score faible signifie que A contient de nombreuses solutions similaires à une autre solution de l'ensemble. Un score élevé signifie que chaque solution est éloignée des autres solutions de l'ensemble.

$$\delta(A) = \frac{1}{|A|} . \sum_{a \in A} \min_{b \in A \setminus \{a\}} d(a, b)$$

Le deuxième indicateur $\delta(A, B)$ décrit la diversité apportée par les solutions présentes dans A mais pas dans B, dans l'union $A \cup B$. Il est égal à la moyenne de l'**apport de diversité** de chaque solution de $A \setminus B$ à $A \cup B$. Un score élevé signifie que chaque solution exclusivement dans A est éloignée des autres solutions de A et B.

$$\delta(A, B) = \frac{1}{|A \setminus B|} . \sum_{a \in A \setminus B} \min_{b \in A \cup B \setminus \{a\}} d(a, b)$$

5.2 Protocole de test

Les tests ont été effectués sur un ordinateur portable HP avec un processeur Intel Core i7-6500U cadencé à 2.50GHz, sous Windows 7 Profesionnel.

Un fichier de 1000 requêtes générées aléatoirement, identique à celui utilisé dans le chapitre 3 a été utilisé pour tester chaque méthode de calcul.

Les calculateurs utilisés pour tester ces algorithmes ont été configurés de manière similaire à la configuration de l'environnement de production :

- rabattements en marche de 1000 mètres maximum autorisés en début et fin de trajet. Un calcul distribué est donc systématiquement effectué (voir chapitre III,

section 3), avec plusieurs arrêts potentiels de départ et de fin, afin d'augmenter la diversité des résultats,

- coefficient multiplicatif de pénalité de correspondance et des rabattement en marche fixé à $C_w = 2$,
- pénalité de changements pour tous les modes fixée à $P_c(M) = 320$ (équivalent secondes)
- durées des transferts minimums de montée et descente dépendantes des modes, $P_t(M) \in \{60, 120, 180\}$ (en secondes).

5.2.1 Instance de test

Les algorithmes de diversité ont été testés sur le réseau R1 uniquement. Nous avons jugé que les autres réseaux ne sont pas assez riches pour nécessiter la mise en place d'un tel outil. Nous avons travaillé sur les mêmes données que les tests de la section 3 du chapitre IV. Pour rappel, le réseau comprend 10 exploitants, 37000 arrêts et 1300 lignes de transport en commun.

5.3 Méthodes monolabels

Nous avons testé six méthodes monolabels en fixant le nombre de solutions souhaitées à trois. Chaque méthode a été exécutée sur chacune des mille requêtes. Les six méthodes sont basées sur l'algorithme 5 présenté précédemment. Le tableau 10 résume les caractéristiques de chaque méthode.

Les deux premières méthodes *intLignes* et *intCourses* utilisent l'interdiction successive de lignes ou de courses. Le paramètre de marge d'interdiction de course, utile uniquement pour *intCourses*, est fixé à $\kappa_1 = 8$ minutes. La condition d'arrêt est d'avoir trouvé les $K = 3$ meilleures solutions. La sélection est ensuite réalisée par la méthode d'élimination ascendante hiérarchique.

La troisième méthode est la même que la première, à la seule différence de la méthode de sélection des trois solutions montrées : KMeans clustering plutôt qu'élimination ascendante hiérarchique.

Les méthodes *finishLabels* et *finishLabelsExt* sont sans recalcul. Elles génèrent toutefois une grande quantité de solutions grâce à la technique de récupération de tous les labels de fin de calcul expliquée dans la section 2.5 du présent chapitre. *FinishLabelsExt* utilise également cette technique et prolonge le calcul. En effet, un calcul s'arrête car les extensions de label sont interdites dès lors qu'elles dépassent le coût du meilleur label trouvé à un nœud de fin. *FinishLabelsExt* autorisent les extensions de label dépassant ce coût, jusqu'à une marge maximale de 2400 (équivalent secondes).

La dernière méthode est un *compromis* entre les interdictions successives de lignes et la récupération des labels de fin : le calcul s'arrête dès lors que six calculs monolabels sont effectués, quelle que soit l'étape à laquelle l'algorithme 5 se trouve. Les ensembles de tous les labels de fin de tous les calculs effectués sont utilisés pour générer des solutions.

Enfin, afin d'obtenir des solutions satisfaisantes du point de vue du critère **lexicographique** (cf. chapitre VI), un calcul graphe *reverse* est effectué sur chacune des $k^{ième}$ meilleure solution (voir algorithme 5), pour toutes les méthodes. Incidemment, cela permet d'augmenter les possibilités de début de trajet pour les méthodes de récupération des labels de fin de calcul. De plus, pour la méthode *compromis*, un calcul graphe *reverse* est effectué sur chaque solution trouvée.

Méthode	**Interdiction**	**Arrêt**	**Graphe *reverse***	**Labels de fin**	**Sélection**
intLignes	lignes	3 best	best	non	EAH
intCourse	courses	3 best	best	non	EAH
KMeans	lignes	3 best	best	non	KMeans
finishLabels	-	1 best	1	oui	EAH
finishLabelsExt	-	1 best	1	oui + 2400	EAH
compromis	lignes	6 calculs	tous	oui + 2400	EAH

Tableau 10 – Méthodes monolabels testées

5.3.1 Résultat

Le tableau 11 montre les moyennes sur les 1000 résultats des indicateurs $\mathtt{i}_{\text{moy}}(\mathsf{A},\mathsf{B})$, $\mathtt{i}_{\max}(\mathsf{A},\mathsf{B})$ de qualité et $\delta(\mathsf{A})$ de diversité des méthodes testées, ainsi que les temps de calculs moyens et les nombres de solutions moyens. B est ici systématiquement *intLignes* et on écrira simplement $\mathtt{i}_{\text{moy}}(\mathsf{A})$, $\mathtt{i}_{\max}(\mathsf{A})$.

Méthode	$\mathtt{i}_{\text{moy}}(\mathsf{A})$ (%)	$\mathtt{i}_{\max}(\mathsf{A})$ (%)	$\delta(\mathsf{A})$ (%)	temps(ms)	nbSol
intLignes	10,7	21,4	45,5	1230	2,94
intCourses	9,1	18,2	39,9	1236	2,94
Kmeans	10,6	21,1	45,0	1250	2,94
finishlabels	8,4	16,3	27,2	342	2,56
finishlabelsExt	11,9	23,5	37,9	335	2,87
compromis	13,7	26,7	52,6	995	2,97

Tableau 11 – Résultats des tests de diversité en monolabel

Analyse du tableau de résultat Il y a quasiment trois solutions en moyenne pour chaque méthode, excepté la méthode des labels de fin, qui n'a que 2.5 solutions en moyenne. En regardant les données brutes, nous constatons qu'il y a de nombreuses occurrences où seules une ou deux solutions sont proposées, contre trois pour les autres méthodes. Cela est largement insuffisant : avant que la qualité ou la diversité des solutions proposées soit prise en compte, il faut avant tout proposer des alternatives si elles existent.

Par ailleurs, les résultats paraissent conformes à ce que l'on pouvait s'imaginer, puisque la méthode d'interdiction successives de lignes, bien que couteuse en temps de calcul, présente des qualités de solution et des indicateurs de diversité parmi les meilleures.

Interdire plutôt les courses augmente la qualité des solutions obtenues, mais diminue de cinq points l'indicateur de diversité, tout en gardant le même temps de calcul. Cette méthode, bien que donnant les meilleures solutions alternatives, est donc inutile du point de vue de la diversité, ce que l'on pouvait bien entendu anticiper.

Il est intéressant de noter que la méthode des *KMeans* ne parait pas significativement différente de l'élimination ascendante hiérarchique, que ce soit en terme de temps de calcul ou de résultat.

La méthode des labels de fin calcule rapidement des alternatives assez bonnes du point de vue de la qualité, mais il y en a trop peu en moyenne, et surtout d'une diversité très médiocre. Son alternative avec l'extension du calcul ne rallonge pas du tout le temps de calcul, mais donne un peu plus de solutions en moyenne, beaucoup plus diversifiée que la première bien que toujours loin du niveau de *intLignes*, et de qualité médiocre.

La méthode de *compromis* a un temps de calcul trois fois plus élevé que *finishLabels*, ce qui est tout à fait naturel puisqu'elle effectue trois fois plus de calculs. Bien que la qualité moyenne et maximale des solutions soient les plus mauvaises, la diversité est bien plus élevée que celle des autres méthodes, et les temps de calcul restent bien en deçà de la méthode *intLignes*.

$\delta(\mathbf{A},\mathbf{B})$ *(%)*	**intL**	**intC**	**KMeans**	**fLabels**	**flExt**	**compr**
intLignes	-	5,8	1,9	10,3	15,8	16,6
intCourses	9,0	-	9,5	10,4	16,0	19,7
Kmeans	2,2	6,6	-	10,1	15,8	17,1
finishLabels	23,1	19,6	22,8	-	12,7	26,3
finishLabelsExt	21,1	18,0	20,9	4,9	-	19,3
compromis	12,8	12,7	13,0	9,7	10,3	-

Tableau 12 – Diversité apportée par la méthode en colonne dans l'ensemble de solutions résultant des méthodes en ligne et colonne

Autres tableaux de résultats La matrice du tableau 12 montre la moyenne de l'indicateur $\delta(\mathsf{A},\mathsf{B})$ de chaque couple de méthodes, avec A la méthode écrite en colonne, et B la méthode écrite en ligne. Par exemple, la méthode *intCourses* apporte 5,8 % de diversité dans l'union des ensembles des résultats *intCourses* et *intLignes*. Ce tableau représente donc en détail la diversité apportée par chaque méthode comparativement à une autre. La supériorité da la méthode *compromis* sur le plan de la diversité est ainsi confirmée : elle apporte beaucoup de diversité à toutes les autres méthodes, mais aucune méthode ne lui apporte beaucoup de diversité. Aussi, la méthode *finishLabels* est clairement inférieure aux autres. Les méthodes d'interdiction successives de lignes ou courses se situent entre les deux.

A==B (%)	**intL**	**intC**	**KMeans**	**fLabels**	**fLExt**	**compr**
intLignes	-	43,2	77,8	4,9	5,0	17,1
intCourses	43,2	-	35,8	5,1	5,1	10,6
Kmeans	77,8	35,8	-	4,8	4,8	15,0
finishLabels	4,9	5,1	4,8	-	32,7	7,9
finishLabelsExt	5,0	5,1	4,8	32,7	-	19,5
compromis	17,1	10,6	15,0	7,9	19,5	-

Tableau 13 – Nombre (en %) des requêtes renvoyant exactement le même ensemble de solutions

La matrice du tableau 13 montre la fraction des cas où les requêtes renvoient exactement le même ensemble de résultats pour les deux méthodes comparées. La première chose que l'on peut noter est que les méthodes *intLignes* et *KMeans*, qui ne diffèrent que par leur méthode de sélection des solutions, présentent plus de 20 % de cas où les ensembles sélectionnés ne sont pas égaux. Les méthodes de *finishLabels* sont très différentes des autres, mais évidemment assez similaires entre elles. Il est intéressant de noter qu'interdire les lignes ou les courses ne change pas les résultats dans plus de 40 % des cas. De manière générale, il y a toujours une fraction non négligeable de requêtes qui ont les mêmes ensembles de solutions, quelles que soient les méthodes comparées.

Histogrammes des indicateurs de qualité et diversité Les histogrammes des figures 35, 36 et 37 présentent la répartition des indicateurs de qualité et diversité de chaque méthode monolabel. Ces histogrammes n'amènent pas beaucoup plus d'information que les moyennes de ces indicateurs. Cela dit, les occurrences où la plus mauvaise solution ($\mathbf{i}_{\max}$) est vraiment mauvaise, c'est-à-dire 30 % pire que la meilleure, sont assez fréquentes pour toutes les méthodes utilisées. Étant donné les méthodes de sélection utilisées dans nos tests, c'était prévisible : l'objectif des agrégations par similarité est bien la diversité et non la qualité des solutions. Une méthode de sélection qui aurait comme effet de diminuer cet effet serait de prendre simplement les trois meilleures solutions, mais on aurait alors des indicateurs de diversité beaucoup plus faibles. La méthode de *compromis* est bien meilleure que les autres en termes de diversité, avec beaucoup d'occurrences d'ensembles à la diversité élevée.

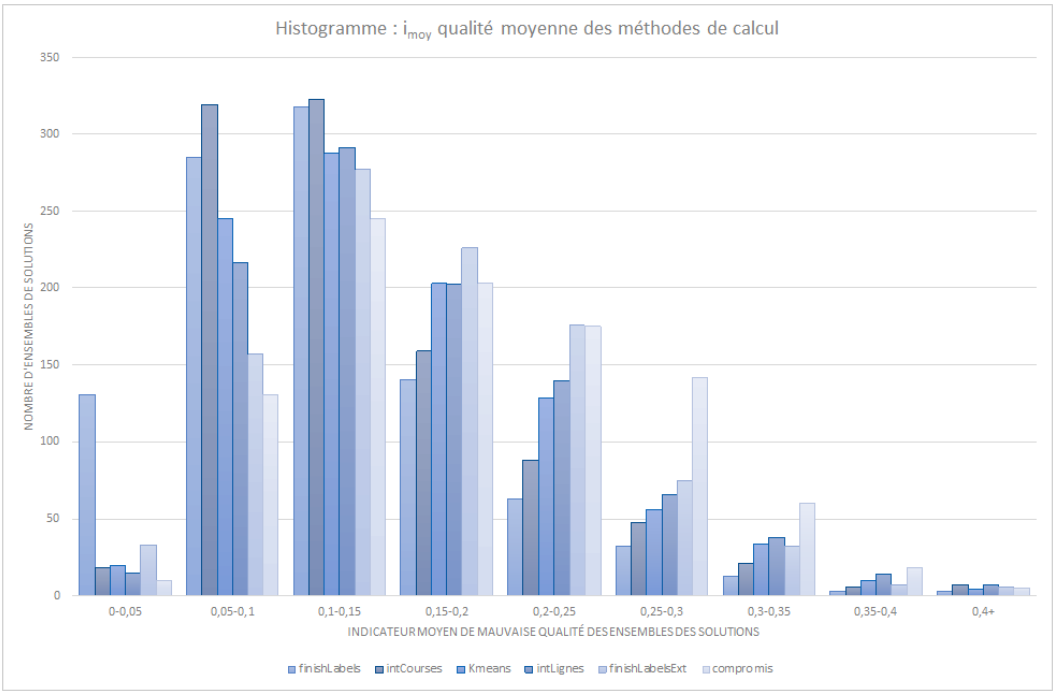


FIGURE 35 – Histogramme : $\mathfrak{i}_{\text{moy}}$ qualité moyenne des ensembles de résultat

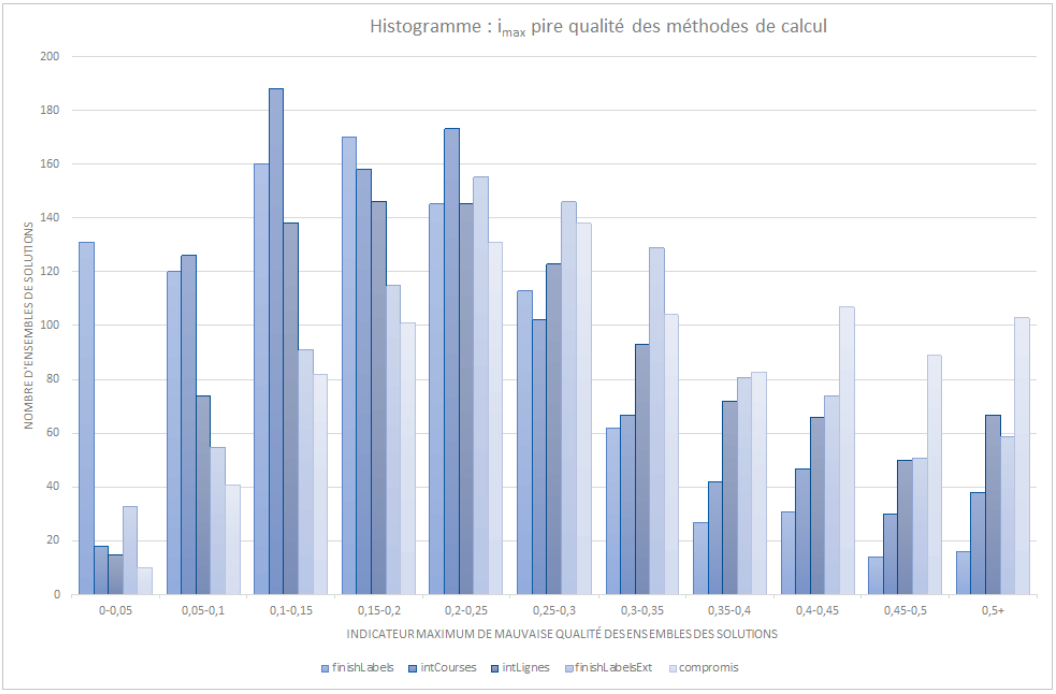


FIGURE 36 – Histogramme : $\mathfrak{i}_{\text{max}}$ pire qualité des ensembles de résultat

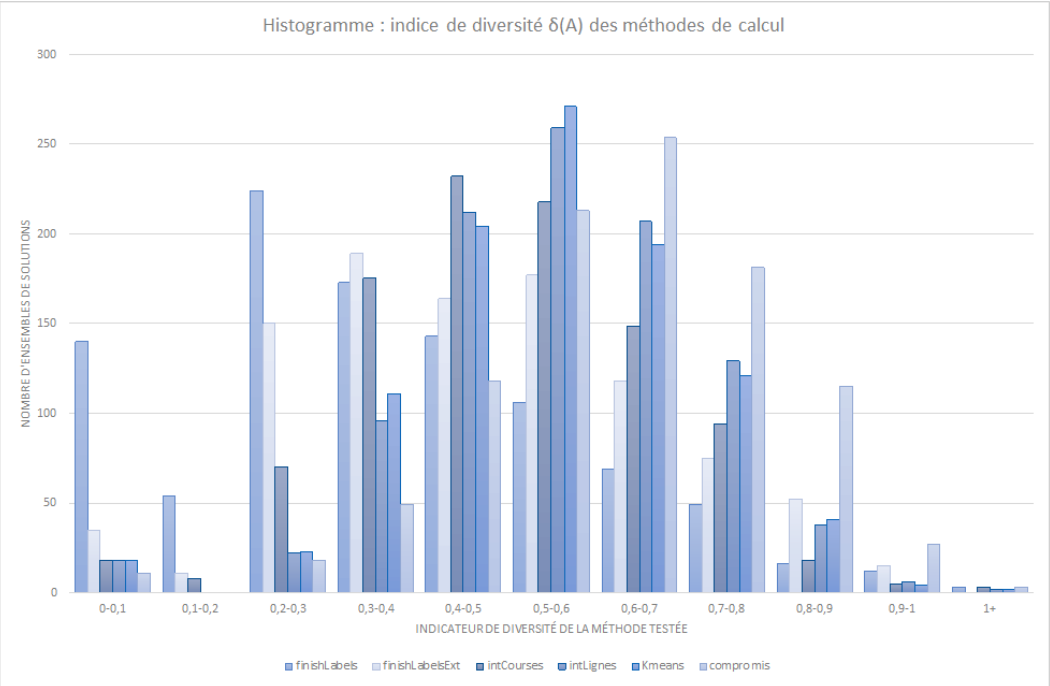


FIGURE 37 – Histogramme : δ diversité des ensembles de résultat

5.4 Méthodes multilabels

Nous avons testé quatre méthodes de génération multilabel en faisant varier uniquement le nombre maximum de labels par nœud : $K = 3$, $K = 6$, $K = 10$ et $K = 15$. Les paramètres de filtrage des labels trop coûteux sont fixés à $\gamma_1 = 1000$ (équivalent secondes) et $\gamma_2 = 1,35$. La méthode de sélection est l'élimination ascendante hiérarchique pour les trois campagnes de test.

Méthode	**K**	γ_1	γ_1	**Graphe *reverse***	**Sélection**
v7 3 labels	3	1000	1,35	non	EAH
v7 6 labels	6	1000	1,35	non	EAH
v7 10 labels	10	1000	1,35	non	EAH
v7 15 labels	15	1000	1,35	non	EAH

Tableau 14 – Méthodes multilabels testées

5.4.1 Résultats

Le tableau 15 montre les moyennes sur les 1000 résultats des indicateurs $i_{moy}(A, B)$, $i_{max}(A, B)$ de qualité et $\delta(A)$ de diversité des méthodes testées, ainsi que les temps de calculs moyen et le nombre de solutions moyen. B est ici systématiquement *intLignes* et on écrira simplement $i_{moy}(A)$, $i_{max}(A)$. Nous montrons également les résultats de quelques méthodes monolabels pour pouvoir les comparer.

La première chose que nous remarquons est qu'une augmentation du nombre de

labels maximum par nœud entraine une assez forte augmentation du temps de calcul. Sur le graphique des temps de calculs 38, nous constatons que les méthodes à dix et quinze labels ont un grand nombre de calculs qui dépassent les trois secondes, ce qui n'est pas acceptable du point de vue de l'utilisateur final.

La qualité moyenne des solutions est correcte, similaire à la qualité des solutions *finishLabelsExt*. Celle-ci ne varie presque pas en fonction de K. Le fait qu'il y ait un ou plusieurs calculs graphe *reverse* en monolabel, alors qu'il n'y en a pas en multilabel, est une explication d'un indicateur de qualité légèrement moins bon en multilabel : le coût d'une solution graphe *reverse* est généralement meilleur que le coût des solutions trouvées en calcul direct, y compris multilabel. Ainsi, il est fréquent (environ une fois sur deux) que la meilleure solution trouvée par une méthode multilabel soit moins bonne que la meilleure solution trouvée par les solutions monolabel.

Le nombre de solutions trouvées est aussi bon qu'avec les méthodes monolabels avec recalcul, et meilleur qu'avec les méthodes monolabels de récupération des labels de fin *finishLabels* et *finishLabelsExt*.

Enfin, même si les indicateurs de diversité sont meilleurs en multilabel que *finishLabelsExt*, seules les versions à dix ou quinze labels par nœud se rapprochent des scores d'*intLignes*. Malgré tout, le tableau 16 montre que les méthodes multilabels apportent environ autant de diversité à la méthode *intLignes* qu'inversement, ce qui indique que chaque type de méthode apporte des solutions différentes. Cela est confirmé par le tableau 17, qui montre que les méthodes multilabels ont de nombreuses instances de résultats strictement identiques entre elles, mais très peu avec les méthodes monolabel.

Méthode	$\mathsf{i}_{\text{moy}}(\mathsf{A})$ (%)	$\mathsf{i}_{\text{max}}(\mathsf{A})$ (%)	$\delta(\mathsf{A})$ (%)	temps(ms)	nbSol
v7 3labels	12,4	22,0	38,5	643	2,94
v7 6labels	12,3	22,0	42,1	1680	2,96
v7 10labels	12,2	21,9	44,4	2684	2,96
v7 15labels	12,2	22,1	46,0	3896	2,96
intLignes	10,7	21,4	45,5	1230	2,94
finishlabelsExt	11,9	23,5	37,9	335	2,87
compromis	13,7	26,7	52,6	995	2,97

Tableau 15 – Résultats des tests de diversité en multilabel

En ce qui concerne les égalités d'ensembles de résultats, le tableau 17 nous montre bien que malgré des temps de calcul croissants, les méthodes multilabel renvoient souvent les mêmes ensembles de solutions. Cela peut s'expliquer par le fait qu'augmenter K a surtout pour effet d'augmenter le nombre de solutions générées, mais la méthode de sélection par élimination ascendante hiérarchique filtre ces solutions pour en extraire les plus diverses, qui se retrouvent souvent être les mêmes. En revanche les solutions renvoyées par les méthodes multilabels sont tout à fait nouvelles par rapport aux solutions des méthodes monolabels.

Enfin les histogrammes 39 et 40 nous montrent respectivement la répartition de l'indice de qualité moyenne et l'indice de diversité des méthodes *intLignes* et multilabel

FIGURE 38 – Temps d'exécution des méthodes 6, 10 et 15 labels par rapport à la méthode à 3 labels par nœud

avec $K = 3$, $K = 6$ et $K = 10$. À nouveau, les méthodes multilabels sont asses proches en terme de qualité. En revanche, la méthode multilabel avec $K = 3$ a beaucoup de cas avec une diversité inférieure à $0,4$, et lorsque $K = 10$, le nombre de cas où la diversité est supérieure à $0,7$ augmente considérablement.

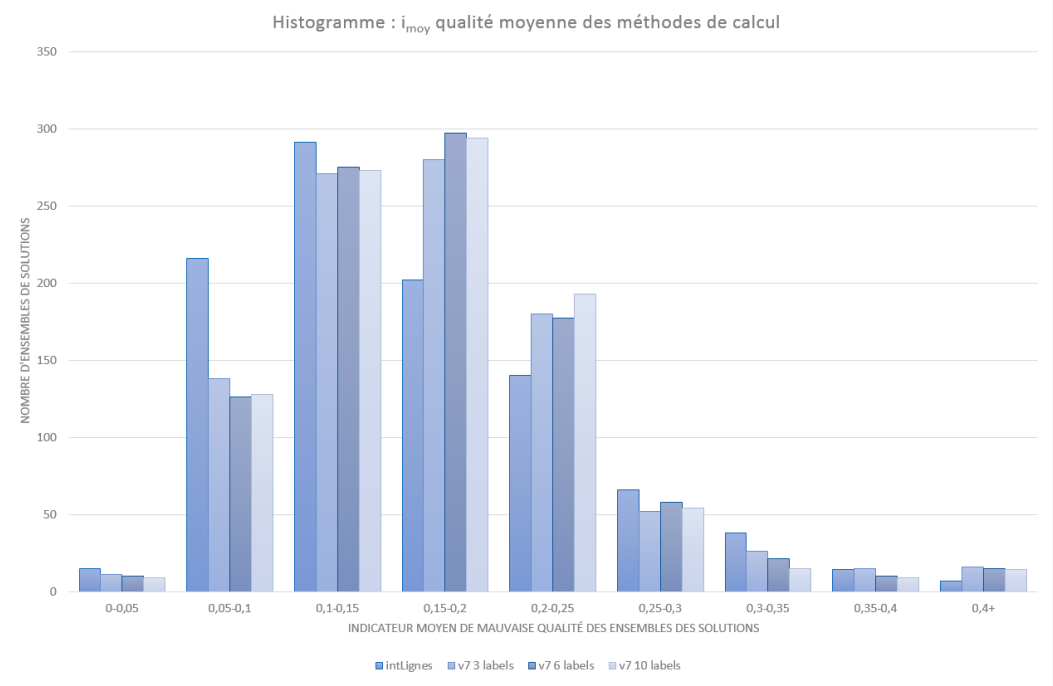


FIGURE 39 – Histogramme : $\mathbf{i}_{\text{moy}}$ qualité moyenne des ensembles multilabels

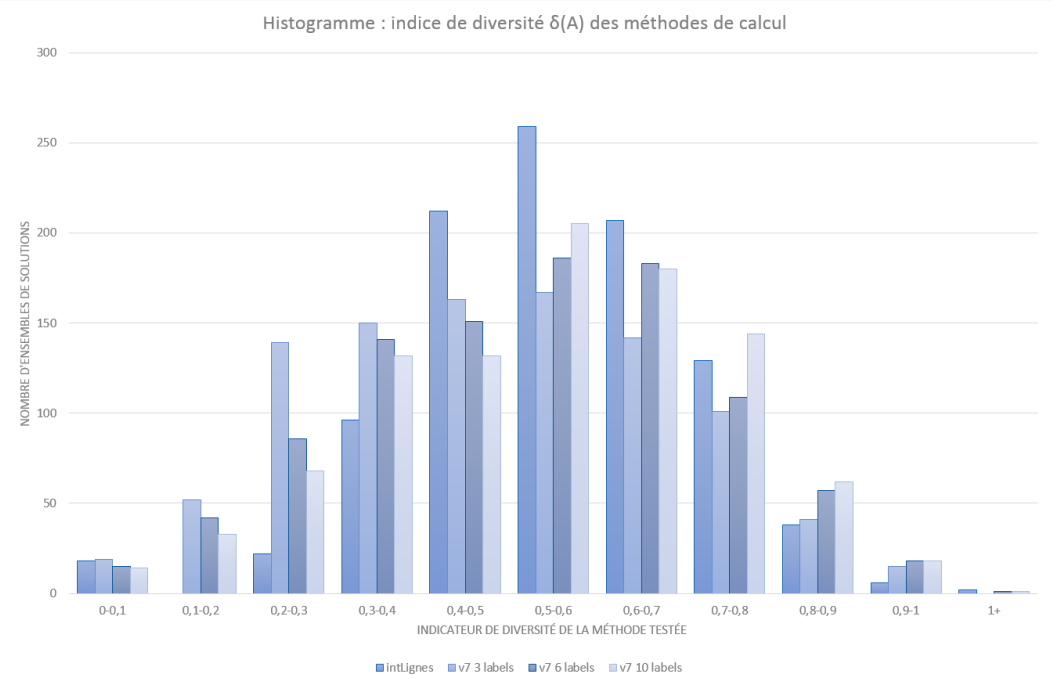


FIGURE 40 – Histogramme : $\boldsymbol{\delta}$ diversité des ensembles multilabels

$\delta(A, B)$ *(%)*	**intL**	**flExt**	**compr**	**3L**	**6L**	**10L**
intLignes	-	15,84	16,57	20,74	22,29	23,09
finishlabelsExt	21,14	-	19,33	21,70	23,86	25,36
compromise	12,84	10,25	-	21,65	22,97	23,84
v7 3 labels	25,45	20,99	29,81	-	9,59	13,52
v7 6 labels	24,76	21,05	28,82	7,47	-	8,24
v7 10 labels	24,26	21,38	28,41	10,21	6,99	-

Tableau 16 – Diversité dans l'union, avec A en colonne et B en ligne

A==B (%)	**intL**	**flExt**	**compr**	**3L**	**6L**	**10L**
intLignes	-	4,95	17,09	5,97	6,17	5,56
finishlabelsExt	4,95	-	19,51	4,75	3,64	2,83
compromise	17,09	19,51	-	5,56	5,16	4,95
v7 3 labels	5,97	4,75	5,56	-	33,06	22,75
v7 6 labels	6,17	3,64	5,16	33,06	-	44,19
v7 10 labels	5,56	2,83	4,95	22,75	44,19	-

Tableau 17 – Nombre (en %) des requêtes renvoyant exactement le même ensemble de solutions

6 Conclusion

Avec des augmentations de temps de calcul très faibles, les résultats des méthodes monolabels sont très encourageants. De plus elles sont faciles à mettre en place dans un environnement de production. Cependant, le paramétrage requis, et les effets agissant sur les solutions présentées sont très complexes, et elle nécessite encore de nombreux essais et aller-retours avec des spécialistes de terrain, qui connaissent les trajets qui valent le coup d'être mis en avant.

Les méthodes multilabels, bien que moins performantes en ce qui concerne les temps de calcul, apportent beaucoup de souplesse dans la manière de traiter le problème. En effet, il est possible d'imaginer de nombreuses évolutions de l'algorithme permettant d'affiner la pertinence des labels conservés. La liberté d'avoir plusieurs labels par nœud et de pouvoir imaginer plusieurs manières de les traiter et de les étendre se fait au prix des performances.

Voici quelques pistes d'améliorations pour l'algorithme multilabel.

- Plutôt qu'utiliser la fonction de hachage, nous pourrions introduire le calcul de distance entre labels, et l'utiliser pour éliminer des labels. Cela ajoute de la complexité à chaque comparaison de labels, mais cela pourrait également réduire le nombre de labels et diminuer le temps de calcul global, tout en augmentant la diversité des résultats.
- Nous pourrions filtrer les labels qui ont plus de N changements supplémentaire à

un autre label, quelle que soit la différence de coût : en effet, s'il existe une solution à un changement, il est peu probable que l'utilisateur préfère une solution à quatre changements, quel que soit le temps gagné.

- Nous pourrions essayer de faire varier les paramètres γ_1 et γ_2, bien que les essais effectués n'aient pas mis en avant de grande différence de temps de calcul.
- Enfin, nous pourrions introduire une étape de calcul graphe *reverse*, avec un horaire d'arrivée bien choisi. En optimisant les calculs pour ne pas doubler des temps déjà longs, il est possible que les méthodes multilabels s'améliorent grandement en qualité et en diversité. Pour cela, nous pouvons nous inspirer des travaux effectués sur le calcul graphe *reverse* en bicritère, expliqués dans le chapitre VI.

Finalement, nous espérons que les travaux contenus dans ce chapitre permettront une évolution parallèle et complémentaire des deux types de méthode, la réflexion sur l'une alimentant les recherches sur l'autre.

Chapitre VI

Critère lexicographique de durée minimale

Sommaire

Ce chapitre traite de la variante du problème de recherche d'itinéraire qui prend en compte le critère **lexicographique de durée minimale**, défini par la définition 28 du chapitre I. Nous traitons dans la section 2 le cas simple de l'optimisation de ce critère en *départ à*. La section 3 est consacrée à la variante lexicographique du problème bicritère **arrivée au plus tôt** et **moins de changements** résolu dans la section 4 du chapitre IV, ainsi qu'à diverses optimisations des temps de calcul. Une dernière section présente les résultats des **expérimentations** faites sur le réseau R1 (présenté dans le chapitre IV) dans le cadre de l'algorithme bicritère lexicographique.

1 Notations et définitions générales

Tous au long de ce chapitre, nous allons exploiter un certain nombre de propriétés liées à l'ensemble des trajets réalisables sur le réseau de transport en commun. Pour cela nous introduisons ici plusieurs notations.

- Notons C le sous-ensemble des trajets réalisables qui partent de A et se terminent à B.
- $\forall \tau \in \Pi$, notons $\mathsf{C}_{\tau,*}$ l'ensemble des trajets de C qui partent après l'horaire τ. C'est l'ensemble des trajets possibles qui respectent $\mathsf{A}@\tau \rightarrow \mathsf{B}$.
- $\forall \tau \in \Pi$, notons $\mathsf{C}_{*,\tau}$ l'ensemble des trajets de C qui arrivent avant l'horaire τ. C'est l'ensemble des trajets possibles qui respectent $\mathsf{A} \rightarrow \mathsf{B}@\tau$.

- $\forall \tau_1 < \tau_2 \in \Pi$, notons C_{τ_1,τ_2} l'ensemble des trajets de C qui partent après l'horaire τ_1 et arrivent avant l'horaire τ_2.

Puisque c'est le sujet du chapitre, rappelons maintenant la définition du critère **lexicographique de durée minimale**. C'est un ordre lexicographique canonique composé de deux critères : le critère d'**arrivée au plus tôt** et le critère de **départ au plus tard**. Si le problème est en *départ à*, l'ordre lexicographique est **arrivée au plus tôt** puis **départ au plus tard**. Si le problème est en *arrivée à*, l'ordre lexicographique est **départ au plus tard** puis **arrivée au plus tôt**. Nous traiterons les problèmes sous leur forme *départ à*.

Rappelons aussi les définitions de solution efficace et de Front de Pareto, déjà vues dans le chapitre IV, afin de rendre plus indépendant ce chapitre pour le lecteur.

Définition 41 *Pour un sous-ensemble $C_1 \subset C$ et un critère $f : C \to \mathbb{N}^k$, ou $\mathbb{N}^k$ est muni d'une relation d'ordre partiel $\prec$ appelée dominance, on dit qu'une solution $S \in C_1$ est **efficace** pour f dans C_1 si*

$$\nexists S' \in C_1, f(S') \prec f(S)$$

Définition 42 *On appelle **ensemble complet de solutions efficaces** pour f dans C_1 et on notera $S_c(C_1, f)$ l'ensemble de toutes les solutions efficaces.*

Définition 43 *On appelle **Front de Pareto** et on note $\mathcal{FP}(C_1, f) \subset \mathbb{N}^k$ l'image directe de $S_c(C_1, f)$ par la fonction f.*

$$\mathcal{FP}(C_1, f) = f(S_c(C_1, f)) = \{f(S) \in C_1 / \nexists S' \in C_1, f(S') \prec f(S)\}$$

Autrement dit, c'est l'ensemble des valeurs optimales du problème d'optimisation du critère f. Pour un critère unique ou s'y ramenant (somme pondérée, lexicographique), cet ensemble ne contient qu'un élément.

Définition 44 *On appelle **ensemble minimal de solutions efficaces** tout ensemble de solutions efficaces de taille minimale, c'est-à-dire tel que chaque valeur du Front de Pareto est atteinte une seule fois par une solution de cet ensemble.*

2 Application à l'algorithme monolabel

Lors d'une requête en *départ à* $A@\tau \to B$ avec comme seul critère l'horaire d'arrivée, l'algorithme monolabel génère une solution partant après τ, arrivant à l'horaire τ_B, optimal selon le critère de l'**arrivée au plus tôt**. Or il existe un ensemble de solutions arrivant exactement à l'horaire τ_B et partant après τ. Cet ensemble peut contenir plus d'un trajet, et le trajet généré par l'algorithme n'est que l'un d'entre eux. Nous cherchons donc un trajet dans cet ensemble qui part le plus tard possible.

Nous exploitons dans cette section un certain nombre de propriétés sur le critère lexicographique afin de présenter un algorithme intuitif qui permet de résoudre ce problème.

2.1 Notations

Nous notons f_1 le critère d'**arrivée au plus tôt** et f_2 le critère de **départ au plus tard**. Pour un trajet $S = A@\tau_A \rightarrow B@\tau_B$, $f_1(T) = \tau_B$ et $f_2(T) = -\tau_A$. Nous notons $Lex_{1,2}$ le critère lexicographique $Lexico(f_1, f_2)$ d'**arrivée au plus tôt** puis **départ au plus tard**. C'est une fonction dans Π^2 muni de la relation d'ordre total lexicographique canonique : pour deux trajets $S_1, S_2 \in C$

$$Lex_{1,2}(S_1) \prec Lex_{1,2}(S_2) \Leftrightarrow f_1(S_1) < f_1(S_2) \vee \big(f_1(S_1) = f_1(S_2) \wedge f_2(S_1) < f_2(S_2)\big)$$

Nous pouvons également remarquer que par définition des critères f_1, f_2 et de l'ensemble C_{t_1,t_2} :

$$S \in C_{t_1,t_2} \Leftrightarrow S \in C \wedge f_1(S) \leq t_2 \wedge f_2(S) \leq -t_1$$

2.2 Propriétés de l'ensemble des solutions efficaces

Nous pouvons tout d'abord remarquer que pour tout critère f à valeurs discrètes et bornées (comme la durée en minutes ou secondes, le nombre de courses empruntées, ...) et sous-ensemble de solutions réalisables, il existe au moins une solution efficace pour f dans ce sous-ensemble.

Proposition 3 *Soit $C_1 \subset C$ non vide. Alors pour tout critère $f : C_1 \rightarrow \mathbb{N}^k$ à valeurs discrètes et bornées, $S_c(C_1, f)$ est non vide.*

Preuve
$f(C_1) \subset \mathbb{N}^k$ est un ensemble fini non vide, puisque les valeurs des critères sont dans des ensembles discrets et bornés. La valeur minimale pour le premier sous-critère f_1 de f est donc atteinte par une solution S, qui ne peut pas être dominée par une autre solution dans C_1. S est donc une solution efficace. Ainsi $S_c(C_1, f) \neq \emptyset$ □

Par définition, toute solution non efficace est dominée par au moins une solution quelconque. Montrons que toute solution non efficace est dominée par au moins une solution efficace.

Proposition 4 *La définition 41 d'une solution efficace dans un ensemble C fini pour un critère f est équivalente à :*

$$S \in S_c(C, f) \Leftrightarrow \forall S' \in S_c(C, f), \neg f(S') \prec f(S)$$

Ce qui équivaut à :

$$S \notin S_c(C, f) \Leftrightarrow \exists S' \in S_c(C, f), f(S') \prec f(S)$$

Preuve
Notons $\mathcal{S} = S_c(C, f)$. Par définition, $S \in C \setminus \mathcal{S} \Leftrightarrow \exists S' \in C, f(S') \prec f(S)$. On montrera :

$$\exists S' \in C, f(S') \prec f(S) \Leftrightarrow \exists S' \in \mathcal{S}, f(S') \prec f(S)$$

Le sens $\Leftarrow$ est trivial, par définition.
Pour montrer le sens $\Rightarrow$, définissons une application $\mathcal{A} : C \mapsto C$ qui à chaque élément S de $C \setminus \mathcal{S}$ associe un élément $S' \in C$ tel que $S' \prec S$, dont on sait l'existence puisque $S \notin \mathcal{S}$. Si $S \in \mathcal{S}$, définissons $\mathcal{A}(S) = S$. Soit $S \in C \setminus \mathcal{S}$.
Supposons par l'absurde que :

$$\begin{aligned}
& \forall i \in \mathbb{N}, \mathcal{A}^i(S) \in C \setminus \mathcal{S} \\
\Rightarrow\ & \forall i \in \mathbb{N}, \mathcal{A}^{i+1}(S) \prec \mathcal{A}^i(S) \\
\Rightarrow\ & \forall i < j \in \mathbb{N}, \mathcal{A}^j(S) \prec \mathcal{A}^i(S) \quad \text{par transitivité de } \prec \\
\Rightarrow\ & \forall i \neq j \in \mathbb{N}, \mathcal{A}^j(S) \neq \mathcal{A}^i(S) \quad \text{car } \prec \text{ est irreflexive}
\end{aligned}$$

On a donc une application injective A_S de $\mathbb{N}$ vers $C \setminus \mathcal{S}$ définie par $A_S(i) = \mathcal{A}^i(S)$. Or $C \setminus \mathcal{S} \subset C$ et C est un ensemble fini. C'est impossible. Finalement, on a :

$$\exists i \in \mathbb{N}, \mathcal{A}^i(S) \in \mathcal{S} \Rightarrow \exists S' \in \mathcal{S}, S' = \mathcal{A}^i(S) \prec \mathcal{A}^{i-1}(S) \prec \cdots \prec S$$

□

Enfin, montrons que trouver une solution efficace pour f_1 arrivant à l'horaire t, puis trouver une solution efficace pour f_2 arrivant avant t permet de trouver une solution efficace pour le critère lexicographique $Lex_{1,2}$.

Proposition 5 *Soit τ tel que $C_{\tau,*}$ est non vide. Soit $S_1 \in S_c(C_{\tau,*}, f_1)$ et $t = f_1(S_1)$, alors il existe une solution $S \in S_c(C_{\tau,t}, f_2)$. De plus, cette solution est efficace dans $C_{\tau,*}$ pour $Lex_{1,2}$.*

Preuve
$C_{\tau,t}$ est non vide puisqu'il contient au moins S_1, qui part après τ et arrive à t. Donc $S_c(C_{\tau,t}, f_2)$ est non vide. Soit S une solution dans $S_c(C_{\tau,t}, f_2)$, on a $f_1(S) \leq t$. Alors :

$$
\begin{aligned}
S \in S_c(C_{\tau,t}, f_2) \quad &\Leftrightarrow S \in C_{\tau,t} \wedge \nexists S' \in C_{\tau,t}, f_2(S') \prec f_2(S) \\
&\Rightarrow f_1(S) \leq t \wedge \nexists S' \in C_{\tau,t}/f_2(S') < f_2(S) \\
&\Rightarrow f_1(S) \leq t \wedge \nexists S' \in C_{\tau,*}/f_1(S') \leq t \wedge f_2(S') < f_2(S) \\
&\Rightarrow \nexists S' \in C_{\tau,*}/f_1(S') \leq f_1(S) \wedge f_2(S') < f_2(S)
\end{aligned}
$$

Or on a également

$$
\begin{aligned}
S_1 \in S_c(C_{\tau,*}, f_1) \quad &\Rightarrow \nexists S' \in C_{\tau,*}, f_1(S') \prec f_1(S_1) \\
&\Rightarrow \nexists S' \in C_{\tau,*}, f_1(S') < f_1(S_1)
\end{aligned}
$$

Et donc

$$
\begin{aligned}
S \in S_c(C_{\tau,t}, f_2) \wedge S_1 \in S_c(C_{\tau,*}, f_1) \\
&\Rightarrow \nexists S' \in C_{\tau,*}/f_1(S') < f_1(S) \vee (f_1(S') \leq f_1(S) \wedge f_2(S') < f_2(S)) \\
&\Rightarrow \nexists S' \in C_{\tau,*}/\mathrm{Lex}_{1,2}(S') \prec \mathrm{Lex}_{1,2}(S) \\
&\Rightarrow S \in S_c(C_{\tau,*}, \mathrm{Lex}_{1,2})
\end{aligned}
$$

□

2.3 Algorithme graphe *reverse* monolabel

L'algorithme utilisé pour résoudre ce problème dans le calcul monolabel découle naturellement de la proposition 5.

Algorithm 10 Algorithme graphe *reverse* monolabel

Trouver une solution S_1 qui minimise f_1 pour la requête en *départ à* $A@\tau \rightarrow B$ en utilisant l'algorithme monolabel 2. $f_1(S_1)$ est l'horaire d'arrivée de S_1.
Trouver une solution S_2 qui minimise f_2 pour la requête en *arrivée à* $A \rightarrow B@f_1(S_1)$, en utilisant l'algorithme 2 sur le graphe graphe *reverse*.

Notons que cela double le temps d'exécution en moyenne, puisqu'un nouveau calcul similaire au premier est effectué sur le graphe TC graphe *reverse*.

Cet algorithme fait partie de l'existant Cityway, à ceci près que le critère f_1 utilisé n'est pas l'horaire d'arrivée, mais la somme pondérée paramétrable agrégeant durée, nombre de changements, marche à pied..., comme décrit dans le chapitre IV. L'algorithme n'est alors plus exact : il se peut que l'horaire de départ du trajet obtenu par l'algorithme soit antérieur à l'horaire de départ demandé, vu que ce n'est pas strictement la durée qui est minimisée.

2.4 Décalage des courses

Un algorithme possible pour trouver une solution S_2 arrivant au même horaire que S_1 est de décaler les courses de S_1 au plus tard possible, sans changer les itinéraires

empruntés. Cet algorithme a pour avantage d'être bien plus rapide que l'algorithme 2, qui parcourt potentiellement tous les nœuds du graphe. Son inconvénient majeur est qu'il n'assure pas de trouver la solution partant au plus tard.

Étant donnée une solution efficace du calcul direct S_1, Soient $I(S_1) = (I_1, \ldots, I_k)$ l'itinéraire emprunté par la solution, et $\mathcal{C}(S_1) = (c_1, \ldots, c_k)$ les courses empruntées par la solution, au sens des définitions données dans la section 2.

Il est alors possible de construire une solution S_2 telle que $C(S_2) = (c'_1, \ldots, c'_k)$, avec $c_k = c'_k$ et $\forall i \in [1, k], I(c_i) = I(c'_i)$, et qui suit la même séquence d'arrêt que S_1 avec l'algorithme 11. LATEST(I, A, h) est une fonction qui permet de trouver la course de l'itinéraire I partant le plus tard possible, mais arrivant à l'arrêt A avant l'horaire h. Nous ne construisons que la séquence de courses et d'horaires, la séquence d'arrêts étant connue. $A_d(c)$ et $h_d(c)$ sont les notations introduites en section 2, et signifient respectivement arrêt de départ et horaire de départ de la course pour le trajet donné.

Algorithm 11 Algorithme de décalage au plus tard

$S_2 = (c_k)$
for $i = k - 1; i \geq 1; i \leftarrow i - 1$ **do**
  $c'_i \leftarrow$ LATEST$(I_i, A_d(c'_{i+1}), h_d(c'_{i+1}))$.
  S_2.addFront(c'_i).
end for
return S_2

Cet algorithme ne garantit pas de trouver la solution qui répond au critère Lexico(f_1, f_2), mais il permet parfois d'améliorer la solution trouvée. Les résultats d'expérimentation montrent que ces solutions ont souvent des horaires de départ antérieurs à ceux des solutions optimales.

3 Application à l'algorithme multilabel bicritère

L'algorithme bicritère décrit dans la section 4 du chapitre IV minimise deux critères : l'**arrivée au plus tôt** et le **nombre de courses empruntées** (égal au **nombre de changements** plus un). Il permet d'obtenir un ensemble de solutions efficaces selon ces critères. De la même manière que pour le cas du monolabel, l'ensemble des solutions efficaces pour ces critères n'est pas le même que l'ensemble des solutions efficaces pour le **critère lexicographique** et le **nombre de courses empruntées**

Nous montrons ici des propriétés qui permettent d'une part de déterminer l'algorithme pertinent pour résoudre le problème, et d'autre part d'optimiser cet algorithme en y ajoutant des règles d'élimination de labels.

3.1 Notations

Nous reprendrons les notations précédentes, en y rajoutant les notations liées au nombre de courses empruntées suivantes.

- f_3 est le critère du **nombre de courses**, que l'on cherche à minimiser. Pour un trajet $S \in C$ tel que $\mathcal{C}(S) = (c_1, \ldots, c_k)$ sont les courses empruntées par S (au sens de la définition 16), $f_3(S) = k$.
- $f_d = (f_1, f_3)$, est le bicritère **horaire d'arrivée** et **nombre de courses** traité dans le chapitre IV (problème en *départ à*).

$$f_d(S_1) \prec f_d(S_2) \Leftrightarrow (f_1(S_1) < f_1(S_2) \wedge f_3(S_1) \leq f_3(S_2)) \vee (f_1(S_1) \leq f_1(S_2) \wedge f_3(S_1) < f_3(S_2))$$

- $f_a(f_2, f_3)$, est le bicritère *arrivée à* : horaire de départ et nombre de courses.

$$f_a(S_1) \prec f_a(S_2) \Leftrightarrow (f_2(S_1) < f_2(S_2) \wedge f_3(S_1) \leq f_3(S_2)) \vee (f_2(S_1) \leq f_2(S_2) \wedge f_3(S_1) < f_3(S_2))$$

- $F_d = (\text{Lex}_{1,2}, f_3)$ est le critère lexicographique et nombre de courses en *départ à*. On cherche à minimiser l'horaire d'arrivée puis à maximiser l'horaire de départ tout en minimisant le nombre de courses.

$$F_d(S_1) \prec F_d(S_2) \Leftrightarrow \begin{cases} \text{Lex}_{1,2}(S_1) \preceq \text{Lex}_{1,2}(S_2) \wedge f_3(S_1) < f_3(S_2) & \vee \\ \text{Lex}_{1,2}(S_1) \prec \text{Lex}_{1,2}(S_2) \wedge f_3(S_1) \leq f_3(S_2) \end{cases}$$

- $\text{Lex}_{2,1}$ est le critère lexicographique $\text{Lexico}(f_2, f_1)$ **départ au plus tard** puis d'**arrivée au plus tôt**.
- $F_a = (\text{Lex}_{2,1}, f_3)$ est le critère lexicographique et nombre de courses en *arrivée à*. On cherche à maximiser l'horaire de départ puis à minimiser l'horaire d'arrivée tout en minimisant le nombre de courses.
- $S = A@\tau_A \underset{n}{\rightarrow} B@\tau_B$ est un trajet S partant de l'arrêt A à l'horaire $f_2(S) = \tau_A$, arrivant à l'arrêt B à l'horaire $f_1(S) = \tau_B$, et empruntant $f_3(S) = n$ courses.

L'algorithme multicritère 4 du chapitre IV permet de déterminer un ensemble minimum de solutions efficaces pour f_d. Notre problématique est de trouver un algorithme permettant de générer un ensemble minimal de solutions efficaces pour F_d dans $C_{\tau,*}$.

Sans perdre de généralités, nous ne traiterons pas le cas du critère F_a dans l'ensemble $C_{*,\tau}$, qui a des propriétés et un algorithme similaire.

3.2 Propriétés sur les ensembles de solutions efficaces pour F_d

Avant de définir l'algorithme, nous allons démontrer plusieurs propriétés des ensembles de solutions recherchés.

Proposition 6 *Pour deux trajets $S_1, S_2 \in C$ et un critère multiple quelconque $f = (f_1, \ldots f_n)$ tel que $f(S_1) \prec f(S_2))$ est défini par $\forall i, f_i(S_1) \leq f_i(S_2)$ et au moins une inégalité stricte, on a :*

$$\neg(f(S_1) \prec f(S_2)) \Leftrightarrow (\exists i / f_i(S_1) > f_i(S_2)) \vee (\forall i, f_i(S_1) = f_i(S_2))$$

C'est à dire que S_1 ne domine pas S_2 selon f si et seulement si ou bien il existe un critère f_i sur lequel S_2 est préférable, ou bien S_1 et S_2 sont équivalentes tous les critères f_i.

PREUVE
Par définition, on a

$$f(S_1) \prec f(S_2) \Leftrightarrow (\exists i / f_i(S_1) < f_i(S_2)) \wedge (\forall i, f_i(S_1) \leq f_i(S_2))$$

et donc

$$\begin{aligned} \neg(f(S_1) \prec f(S_2)) &\Leftrightarrow \neg(\exists i / f_i(S_1) < f_i(S_2)) \vee \neg(\forall i, f_i(S_1) \leq f_i(S_2)) \\ &\Leftrightarrow (\forall i, f_i(S_1) \geq f_i(S_2)) \vee (\exists i / f_i(S_1) > f_i(S_2)) \end{aligned}$$

Or la proposition $\forall i, f_i(S_1) \geq f_i(S_2)$ équivaut à

$$\begin{aligned} &\forall i, f_i(S_1) > f_i(S_2) \vee f_i(S_1) = f_i(S_2) \\ &\Leftrightarrow (\forall i, f_i(S_1) = f_i(S_2)) \vee (\exists i / f_i(S_1) > f_i(S_2) \wedge \forall i, f_i(S_1) \geq f_i(S_2)) \end{aligned}$$

Et donc en réinjectant cette expression dans la première équation et en factorisant $(\exists i / f_i(S_1) > f_i(S_2))$, on a bien :

$$\neg(f(S_1) \prec f(S_2)) \Leftrightarrow (\exists i / f_i(S_1) > f_i(S_2)) \vee (\forall i, f_i(S_1) = f_i(S_2))$$

□

Lemme 1 *Pour deux trajets $S_1, S_2 \in C$ et le critère F_d, on déduit de la proposition 6 le lemme suivant.*

$$\neg(F_d(S_1) \prec F_d(S_2)) \Leftrightarrow \begin{cases} f_3(S_1) > f_3(S_2) & \vee \\ f_1(S_1) > f_1(S_2) & \vee \\ f_1(S_1) = f_1(S_2) \wedge f_2(S_1) > f_2(S_2) & \vee \\ f_3(S_1) = f_3(S_2) \wedge f_1(S_1) = f_1(S_2) \wedge f_2(S_1) = f_2(S_2) & \end{cases}$$

PREUVE

$$\neg(F_d(S_1) \prec F_d(S_2)) \Leftrightarrow \begin{cases} f_3(S_1) > f_3(S_2) & \vee \\ Lex_{1,2}(S_1) > Lex_{1,2}(S_2) & \vee \\ f_3(S_1) = f_3(S_2) \wedge Lex_{1,2}(S_1) = Lex_{1,2}(S_2) & \end{cases}$$

Vu $Lex_{1,2}(S_1) > Lex_{1,2}(S_2) \Leftrightarrow (f_1(S_1) > f_1(S_2) \vee (f_1(S_1) = f_1(S_2) \wedge f_2(S_1) > f_2(S_2)))$
et $Lex_{1,2}(S_1) = Lex_{1,2}(S_2) \Leftrightarrow (f_1(S_1) = f_1(S_2) \wedge f_2(S_1) = f_2(S_2))$,
on a bien le résultat établi. □

Proposition 7 *Les solutions efficaces pour* F_d *ne sont pas toutes efficaces pour* f_d*. Les solutions efficaces pour* f_d *ne sont pas toutes efficaces pour* F_d*, c'est-à-dire :*

$$\exists C_1 \subset C, S_c(C_1, F_d) \not\subset S_c(C_1, f_d) \wedge S_c(C_1, f_d) \not\subset S_c(C_1, F_d)$$

Preuve

Contre exemple : supposons C_1 composé des trois trajets suivants.

$A@08:00 \underset{1}{\rightarrow} B@10:00$	$\notin S_c(C_1, F_d)$	$\in S_c(C_1, f_d)$
$A@08:15 \underset{1}{\rightarrow} B@10:00$	$\in S_c(C_1, F_d)$	$\in S_c(C_1, f_d)$
$A@08:30 \underset{2}{\rightarrow} B@10:00$	$\in S_c(C_1, F_d)$	$\notin S_c(C_1, f_d)$

□

Proposition 8 *Les horaires d'arrivées optimaux pour* F_d *sont les mêmes que pour* f_d*, c'est-à-dire :*

$$\forall C_1 \subset C, f_1(S_c(C_1, F_d)) = f_1(S_c(C_1, f_d))$$

Preuve

Pour faire cette démonstration, notons tout d'abord l'équivalence suivante :

$$\forall S_1, S_2 \in C_1 / f_1(S_1) \neq f_1(S_2)$$
$$f_d(S_1) \prec f_d(S_2) \Leftrightarrow F_d(S_1) \prec F_d(S_2)$$

On le montre directement en écrivant les définitions de F_d et f_d et en se servant du fait que si on suppose $f_1(S_1) \neq f_1(S_2)$, on a :

$$f_1(S_1) \leq f_1(S_2) \Leftrightarrow f_1(S_1) < f_1(S_2) \Leftrightarrow Lex_{1,2}(S_1) \prec Lex_{1,2}(S_2) \Leftrightarrow Lex_{1,2}(S_1) \preceq Lex_{1,2}(S_2)$$

Montrons par l'absurde que $f_1(S_c(C_1, F_d)) \subset f_1(S_c(C_1, f_d))$ en supposant :

$$\exists \tau \in \Pi / \tau \in f_1(S_c(C_1, F_d)) \wedge \tau \notin f_1(S_c(C_1, f_d))$$
$$\Leftrightarrow \exists S \in S_c(C_1, F_d) \wedge \forall S' \in S_c(C_1, f_d), f_1(S) \neq f_1(S')$$

Soit S une telle solution. On sait que $S \in C_1 \setminus S_c(C_1, f_d)$, sinon on aurait : $S' = S \Rightarrow f_1(S') = f_1(S)$.
Donc vu la proposition 4, $\exists S' \in S_c(C_1, f_d) / f_d(S') \prec f_d(S)$
Mais comme $f_1(S) \neq f_1(S')$, $f_d(S') \prec f_d(S) \Leftrightarrow F_d(S') \prec F_d(S)$, ce qui est en contradiction avec $S \in S_c(C_1, F_d)$.
On peut donc en conclure que $f_1(S_c(C_1, F_d)) \subset f_1(S_c(C_1, f_d))$. Pour l'inclusion inverse, le raisonnement est identique. □

Proposition 9 *Soit* $\mathcal{S} = (S_i)_{i \in [1,k]}$ *un ensemble minimal de solutions efficaces pour* f_d *dans* $C_{\tau,*}$*, avec les horaires d'arrivée* $t_i = f_1(S_i)$ *strictement croissants. On peut partitionner* $S_c(C_{\tau,*}, F_d)$ *en sous-ensembles définis par :*

$$\mathcal{S}_i = \{S \in S_c(C_{\tau,*}, F_d) / f_1(S) = t_i\}$$

PREUVE
Les partitions sont disjointes : $\forall i \neq j \in [1,k], f_1(S_i) \neq f_1(S_j)$ car la série des $f_1(S_i)$ est strictement croissante. $\mathcal{S}_i \cap \mathcal{S}_j = \{S / f_1(S) = f_1(S_i) \wedge f_1(S) = f_1(S_j)\} = \emptyset$
Les partitions sont non vides : vu la proposition 8, pour tout horaire d'arrivée $f_1(S_i)$, il existe une solution de $S_c(C_{\tau,*}, F_d)$ arrivant au même horaire.
Les partitions sont couvrantes : vu la proposition 8, toute solution $S \in S_c(C_{\tau,*}, F_d)$ arrive à un horaire $f_1(S) \in f_1(S_c(C_{\tau,*}, f_d) = \{f_1(S_i)\}$
□

Proposition 10 *Le nombre de courses de chaque solution dans chacune des partitions $\mathcal{S}_i$ est borné par les $n_i = f_3(S_i)$.*

$$\forall i \in [1,k], \forall S \in \mathcal{S}_i, f_3(S) \geq n_i$$
$$\forall i \in [2,k], \forall S \in \mathcal{S}_i, f_3(S) < n_{i-1}$$

PREUVE
Supposons par l'absurde qu'il existe i tel que $f_3(S_i) > f_3(S)$, alors vu que $f_1(S_i) = f_1(S)$, on aurait $f_d(S) \prec f_d(S_i)$, ce qui est absurde.
Supposons maintenant qu'il existe i tel que $f_3(S) \geq f_3(S_{i-1)}$. On sait que $f_1(S_{i-1}) < f_1(S_i) = f_1(S)$, donc $Lex_{1,2}(S_{i-1}) \prec Lex_{1,2}(S)$ et donc $F_d(S_{i-1}) \prec F_d(S))$, ce qui est absurde □

Proposition 11 *Soit $\{\mathcal{S}_i\}_{i\in[1,k]}$ les ensembles partitionnant $S_c(C_{\tau,*}, F_d)$ de la proposition 9. Soit $C_i \subset C_{\tau,*}$ l'ensemble des trajets partant après τ, arrivant exactement à t_i et ayant strictement moins de n_{i-1} courses si $i > 1$. Pour tout $i \in [1,k]$, soit $\mathcal{S}'_i = S_c(C_i, f_a)$ les solutions efficaces pour f_a dans C_i.*

$$\mathcal{S}_i = \mathcal{S}'_i$$

PREUVE
Soit $i \in [1,k]$. On remarquera que si l'on suppose S_1, S_2 tels que $f_1(S_1) = f_1(S_2)$, on a $Lex_{1,2}(S_1) \prec Lex_{1,2}(S_2) \Leftrightarrow f_2(S_1) \prec f_2(S_2)$, ce qui implique :

$$F_d(S_1) \prec F_d(S_2) \Leftrightarrow f_a(S_1) \prec f_a(S_2)$$

Montrons $\mathcal{S}_i \subset \mathcal{S}'_i$. Supposons par l'absurde $S \in \mathcal{S}_i$ et $S \notin \mathcal{S}'_i$. On a donc une solution S' optimale pour f_a dans C_i : $f_a(S') \prec f_a(S)$. Or $f_1(S) = f_1(S')$, donc $F_d(S') \prec F_d(S)$, ce qui est contradictoire avec $S \in \mathcal{S}_i$.

Montrons $\mathcal{S}'_i \subset \mathcal{S}_i$. Supposons par l'absurde $S \in \mathcal{S}'_i$ et $S \notin \mathcal{S}_i$. On a donc une solution S' optimale pour F_d dans $C_{\tau,*}$ tq $F_d(S') \prec F_d(S)$. On sait alors que S' est dans un des $\mathcal{S}_i$.

Ou bien $f_1(S') = t_i = f_1(S)$ et $F_d(S') \prec F_d(S)$, donc $f_a(S') \prec f_a(S)$, ce qui est contradictoire avec $S \in \mathcal{S}'_i$. Ou bien $S' \in \mathcal{S}_j$ avec $j < i$, et donc $f_3(S') = n_j \geq n_{i-1} > f_3(S)$, ce qui est contradictoire avec $F_d(S') \prec F_d(S)$. Ou bien $S' \in \mathcal{S}_j$ avec $j > i$ et $f_1(S') > f_1(S)$ ce qui est contradictoire avec $F_d(S') \prec F_d(S)$. □

On peut conclure de cette dernière propriété que si l'on trouve un ensemble minimal de solutions efficaces pour f_a dans chaque ensemble C_i, l'union de ces solutions est un ensemble minimal de solutions efficaces pour F_d dans $C_{\tau,*}$.

3.3 Algorithme bicritère graphe *reverse*

L'algorithme bicritère graphe *reverse* découle naturellement de la proposition 11. On note $S_m(f, C)$ un ensemble minimal de solutions efficaces pour un critère f dans un ensemble C, que l'on sait calculer si f est f_d ou f_a. L'algorithme calcule d'abord l'ensemble $S_d = (S_1, \ldots, S_k) \leftarrow S_m(f_d, C_{\tau,*})$ avec l'algorithme multilabel 4 appliqué au critère f_d. Puis pour chaque solution S_i et ensemble C_i des trajets partant après τ, arrivant exactement à t_i et ayant strictement moins de n_{i-1} changements si $i > 1$, il calcule l'ensemble $S_m(f_a, C_i)$ et l'ajoute à l'ensemble de solutions efficaces pour F_d. Les contraintes définissant C_i modifient légèrement l'algorithme en interdisant les labels qui ne les respectent pas lors de leur extension.

Notons également que la solution S_i ne peut pas dominer les solutions cherchées (il faut supprimer le premier temps d'attente de S_i pour rendre cette condition plus restrictive, sans quoi limiter l'horaire de départ à τ est une restriction plus forte).

Nous avons donc défini l'algorithme 12 bicritère lexicographique, nombre de changements ci-dessous.

Algorithm 12 Algorithme bicritère graphe *reverse*

$S \leftarrow$ liste vide
$S_d = (S_1, \ldots, S_k) \leftarrow S_m(f_d, C_{\tau,*})$ triées par horaire d'arrivée croissant
for all $i \in [1, k]$ **do**
  $C_i \leftarrow$ défini par les contraintes : départ après τ, arrivée exactement à $f_1(S_i)$, si $i > 1$ moins de courses que $f_3(S_{i-1})$, non dominé par S_i
  $S \leftarrow S \cup S_m(f_a, C_i)$
end for
return S

Ainsi, lorsqu'il y a plusieurs solutions avec des horaires d'arrivée t_i différents dans le résultat du calcul bicritère direct, il faut effectuer le calcul multilabel en graphe *reverse* pour chaque t_i. Cela multiplie le temps de calcul d'autant et chaque calcul est plus couteux en temps CPU qu'un calcul monolabel. Par conséquent, il nous a paru nécessaire de rechercher des optimisations permettant de réduire ces temps de calculs.

3.4 Exemple

Donnons un exemple théorique de ce qu'apporte l'algorithme graphe *reverse* pour trois solutions, avec comme arrêt de départ la Gare, arrêt d'arrivée la Mairie, et heure de départ 08:00.

Supposons que les trois trajets suivants résultent d'un calcul bicritère en *départ à*, ensemble minimal de solutions pour $\mathtt{f_d}$.

$$\begin{array}{l} \text{Gare@08:15} \underset{4}{\rightarrow} \text{Mairie@09:50} \\ \text{Gare@08:05} \underset{2}{\rightarrow} \text{Mairie@10:30} \\ \text{Gare@13:00} \underset{1}{\rightarrow} \text{Mairie@14:00} \end{array} \quad \text{avec} \begin{cases} 4>2>1 \\ 09{:}50<10{:}30<14{:}00 \end{cases}$$

Voici maintenant un ensemble minimal de solutions pour $\mathtt{F_d}$, que l'on peut trouver avec l'algorithme 12.

$$\begin{array}{lcl} 08{:}00 \underset{4}{\rightarrow} 09{:}50 & \xrightarrow{S \rightarrow A@09:50} & \begin{array}{l} 08{:}20 \underset{5}{\rightarrow} 09{:}50 \\ 08{:}15 \underset{4}{\rightarrow} 09{:}50 \end{array} \\ 08{:}00 \underset{2}{\rightarrow} 10{:}30 & \xrightarrow{S \rightarrow A@10:30} & \begin{array}{l} 09{:}30 \underset{3}{\rightarrow} 10{:}30 \\ 09{:}00 \underset{2}{\rightarrow} 10{:}30 \end{array} \\ 08{:}00 \underset{1}{\rightarrow} 14{:}00 & \xrightarrow{S \rightarrow A@14:00} & 13{:}30 \underset{1}{\rightarrow} 14{:}00 \end{array}$$

Dans cet exemple, les solutions à cinq changements et à trois changements n'apparaissent pas du tout dans le calcul direct, puisque leurs horaires d'arrivée n'est pas meilleur que ceux des solutions à quatre et deux changements respectivement. De plus, la solution à deux changements est meilleure car elle dure 55 minutes de moins.

Pour les obtenir, il faut résoudre les trois requêtes en *arrivée à* suivantes, avec les contraintes sur les labels associées :

- Gare→Mairie, arrivée exactement à 09:50, horaire≥08:00, n'est pas dominé par (08:15,4) selon $\mathtt{f_a}$.
- Gare→Mairie, arrivée exactement à 10:30, horaire≥08:00, courses<4 (2 ou 3), n'est pas dominé par (08:05,2) selon $\mathtt{f_a}$.
- Gare→Mairie, arrivée exactement à 14:00, horaire≥08:00, courses<2 (1), n'est pas dominé par (13:00,1) selon $\mathtt{f_a}$.

3.5 Composition de trajet et valeurs des critères

Dans le graphe TC *time-dependant*, un trajet S de A vers B, partant après τ est une succession de couple nœuds du graphe, horaire. Chaque couple de nœuds successifs étant

présent parmi les arcs du graphe et respecte un certain nombre de règles, énoncées dans les descriptions des algorithmes monolabel et multilabel. Pour tout nœud intermédiaire N, on peut décomposer le trajet S en deux sous-trajets S_N et S'_N allant respectivement de A vers N et de N vers B, qui respectent certaines propriétés.

Proposition 12 *Soit* $S \in C_{\tau,*}$, *décomposé en* $S_N : A \to N$ *et* $S'_N : N \to B$. *On a :*

- $f_1(S) = f_1(S'_N) > f_1(S_N)$.
- $f_2(S'_N) = -f_1(S_N)$
- $f_3(S) \geq f_3(S_N)$.

PREUVE
L'horaire d'arrivée du trajet S est le même que la deuxième et dernière partie de la décomposition S'_N. De plus, chaque arc étant parcouru en un temps non nul, on cet horaire est strictement supérieur à l'horaire d'arrivée à N.

L'horaire de départ de S'_N est égal à l'horaire d'arrivée de S_N par définition de la décomposition de S en S_N et S'_N.

Le nombre de courses empruntées par le trajet total est évidemment supérieur ou égal au nombre de courses empruntées par chaque sous-trajet. □

Proposition 13 *Soit* $S \in C_{\tau,*}$, *décomposé en* $S_N : A \to N$ *et* $S'_N : N \to B$. *On a :*

- *Si* N *est un nœud* MARCHE *:* $f_3(S) = f_3(S_N) + f_3(S'_N)$.
- *Si* N *est un nœud* ITI *:* $f_3(S) = f_3(S_N) + f_3(S'_N) - 1$.

PREUVE
Si N est un nœud MARCHE, N est un arrêt de changement de S et le nombre de courses empruntées par S est la somme des deux sous-trajets.

Si N est un nœud ITI, N est un arrêt sur itinéraire. Soit C le premier nœud MARCHE antérieur à N et D le premier nœud MARCHE postérieur à N. Il y a exactement un arc GETIN entre C et N et exactement un arc GETOFF entre N et D. En décomposant S en trois sous trajets S_1, S_2, S_3 de A vers C, de C vers D et de D vers B, on a $f_3(S) = f_3(S_1) + f_3(S_2) + f_3(S_3) = f_3(S_1) + 1 + f_3(S_3)$. Or $f_3(S_N) = f_3(S_1) + 1$ et $f_3(S'_N) = f_3(S_3) + 1$. □

Notons que f_3, le nombre de courses empruntées par un trajet d'un nœud MARCHE à un autre nœud MARCHE, est égal au nombre d'arcs GETIN ou GETOFF qu'il contient. La valeur de f_3 pour un trajet d'un nœud MARCHE à un nœud ITI est égal au nombre d'arcs GETIN. La valeur de f_3 pour un trajet d'un nœud ITI à un nœud MARCHE est égal au nombre d'arcs GETOFF.

3.6 Propriétés sur les nœuds intermédiaires

Nous verrons dans cette partie comment la conservation d'une liste de labels efficaces pour f_d lors du calcul direct peut se révéler utile pour accélérer les calculs graphe *reverse* optimisant f_a.

3.6.1 Hypothèses

Dans les propositions suivantes, N est un nœud du graphe TC différent de A et de B. L'algorithme multilabel 4 optimisant f_d dans $C_{\tau,*}$ établit un ensemble de chemins efficaces S_N dont les valeurs pour (f_1, f_3) sont notées $\{(t_j, m_j)\}_{j\in[1,l]}$ avec les t_j croissants et les m_j décroissants. Cet ensemble peut être vide.

On suppose qu'on a déterminé un ensemble C_i de trajets parmi lesquels on cherche à optimiser f_a. Pour rappel C_i est l'ensemble des trajets de A vers B, partant après τ, arrivant exactement à une date t_i, et, si $i > 1$, ayant strictement moins de n_{i-1} courses.

On suppose également que l'algorithme graphe *reverse* cherchant à optimiser f_a est en cours d'exécution, et a établi un trajet S'_N allant de N vers B , partant à une date τ_R et comprenant n_R courses (nombre d'arcs GETOFF).

3.6.2 Propositions

On se pose la question de savoir sous quelles conditions il est possible d'étendre le trajet S'_N pour obtenir une solution efficace pour f_a dans C_i et efficace pour F_d dans $C_{\tau,*}$.

Proposition 14 *Si $S_N = \emptyset$, alors aucune solution efficace pour F_d dans $C_{\tau,*}$ ne passe par N. On peut donc éliminer le label.*

PREUVE
Supposons par l'absurde qu'il existe un trajet efficace S pour F_d dans $C_{\tau,*}$ et qui contient le nœud N. On peut diviser ce chemin en deux parties : S_N qui va de A vers N et S'_N de N vers B. Vu la proposition 12, on a : $f_1(S) > f_1(S_N)$ et $f_3(S) \geq f_3(S_N)$.

L'algorithme 4 s'arrête lorsque le tas est vide. Hors le tas se vide quand les nouveaux labels créés sont dominés par un label final, ou quand il n'y a plus de chemin à explorer. Ce dernier cas est impossible puisqu'il existe un trajet de A vers B qui contient le nœud N et qui part après τ. On sait donc qu'il existe un trajet efficace S_e de A vers B qui domine S_N selon f_d. On a $f_d(S_e) \prec f_d(S_N)$, ce qui implique :

$$f_1(S_e) \leq f_1(S_N) \wedge f_3(S_e) \leq f_3(S_N)$$

Or $f_1(S) > f_1(S_N)$ et $f_3(S) \geq f_3(S_N)$, donc

$$f_1(S_e) > f_1(S) \wedge f_3(S_e) \geq f_3(S)$$

Ce qui implique finalement :

$$F_d(S_e) \prec F_d(S)$$

Ce qui est en contradiction avec S efficace pour F_d. □

Proposition 15 *On suppose ici* $i > 1$ *et on note* $n_B = n_{i-1}$ *la borne de nombre de courses. On suppose également* $\mathcal{S}_N \neq \emptyset$*, on peut noter* $n = \min(m_j)$ *le nombre de courses minimum trouvé lors du calcul direct au nœud* N*.*

Si N *est un nœud* MARCHE*, et :*

$$n + n_R \geq n_B$$

ou, si N *est un nœud* ITI*, et :*

$$n + n_R - 1 \geq n_B$$

alors il n'existe aucune solution dans C_i *qui soit le prolongement de* S'_N*, et on peut éliminer ce label.*

PREUVE

Montrons la contraposée : soit S une solution dans C_i qui soit le prolongement de S'_N. Notons S_N la partie de S entre A et N. On ne peut pas avoir de solution entre A et N partant après τ en strictement moins de n courses. On a donc $f_3(S_N) \geq n$. De plus S dans C_i implique $f_3(S) < n_B$.

Si N est un nœud MARCHE, vu la proposition 13, on a $f_3(S) = f_3(S_N) + f_3(S'_N)$:

$$n_B > f_3(S) = f_3(S_N) + f_3(S'_N) = f_3(S_N) + n_R \geq n + n_R$$

. Si N est un nœud ITI, vu la proposition 13, on a $f_3(S) = f_3(S_N) + f_3(S'_N) - 1$:

$$n_B > f_3(S) = f_3(S_N) + f_3(S'_N) - 1 = f_3(S_N) + n_R - 1 \geq n + n_R - 1$$

. □

Proposition 16 *On suppose toujours* $i > 1$ *et on note* $n_B = n_{i-1}$ *la borne de nombre de courses. On suppose également* $S_N \neq \emptyset$*, ainsi que* $\exists j / m_j + n_R < n_B$ *si* N *est un nœud* MARCHE*. Si* N *est un nœud* ITI*, on suppose plutôt* $\exists j / m_j + n_R - 1 < n_B$*.*

Si N *est un nœud* MARCHE*, et :*

$$\min_{j/m_j+n_R<n_B} (t_j) > \tau_R$$

ou, si N *est un nœud* ITI*, et :*

$$\min_{j/m_j+n_R-1<n_B} (t_j) > \tau_R$$

alors il n'existe aucune solution dans C_i *qui soit le prolongement de* S'_N*, et on peut éliminer ce label.*

PREUVE

On ne démontrera que pour le cas où N est un nœud MARCHE, la démonstration étant identique pour le cas où N est un nœud ITI.

Montrons par contraposée que l'existence d'une solution S dans C_i qui soit le prolongement de S'_N implique :

$$\min_{j/m_j<n_B-n_R} (t_j) \leq \tau_R$$

On a supposé que $\exists j / m_j + n_R < n_B$, donc le min existe est il est atteint.

Notons S_N la partie de S entre A et N.

$$\begin{aligned}
\forall j, \neg f_d(S_N) &\prec (t_j, m_j) \\
\text{donc } \forall j, \neg(f_1(S_N), f_3(S_N)) &\prec (t_j, m_j) \\
\text{donc } \forall j, \neg(\tau_R, f_3(S) - n_R) &\prec (t_j, m_j) \\
\text{donc } \forall j, \tau_R \geq t_j \vee f_3(S) - n_R &\geq m_j) \\
\text{or on a } f_3(S) < n_B, \text{ donc} & \\
\text{donc } \forall j, (\tau_R \geq t_j) \vee (n_B - n_R &\geq m_j) \\
\text{donc } \forall j, (n_B - n_R < m_j) &\Rightarrow (t_j \leq \tau_R) \\
\text{et donc } \min_{j/m_j<n_B-n_R} (t_j) &\leq \tau_R
\end{aligned}$$

□

3.6.3 Optimisations

Les propositions 14, 15 et 16 nous donnent donc trois critères d'élimination des labels :

- s'ils n'ont pas été atteint lors de l'algorithme direct,
- s'ils ont été atteint mais en un nombre de changements minimum trop grand,
- s'ils ont été atteint en un temps minimum trop grand.

Ils modifient légèrement l'algorithme 12, en particulier la partie qui génère les trajets efficaces $\mathsf{S}_\mathsf{m}(\mathsf{f}_\mathsf{a}, \mathsf{C}_\mathsf{i})$, grâce à l'algorithme 4 graphe *reverse*. Lors de l'extension d'un label, chaque type d'élimination potentielle est vérifiée en fonction des informations obtenues lors du calcul direct, et le label n'est pas étendu si il correspond à l'un des critères d'élimination.

4 Expérimentations

Le calcul du graphe *reverse* n'a été testé que sur l'instance de test de R1, la plus importante en terme de nombres de solutions et de temps de calcul.

Le tableau 18 donne les temps moyen d'exécution des calculs directs, graphe *reverse* non optimisé et optimisé, sur les 1000 requêtes aléatoires. La figure 41 nous montre les même données sous forme d'un graphique avec en abscisse le temps de référence du calcul direct et en ordonnée les temps d'exécution de la même requête en graphe *reverse* et en graphe *reverse* optimisé.

Méthode	Moyenne (ms)	Max. (ms)
Direct	275	722
Décalage	295	780
Graphe *reverse*	687	2626
Graphe *reverse* optimisé	419	1134

Tableau 18 – Temps d'exécution

Le tableau 19 compare la solution de décalage des courses et la solution du calcul du graphe *reverse* exact. Le pourcentage d'amélioration correspond à la proportion de solutions dont les horaires de départ ont pu être avancés pour arriver au même horaire d'arrivée avec le même nombre de changements. Le temps gagné est la moyenne de la différence entre l'horaire de départ demandé et l'horaire de départ trouvé par le calcul (graphe *reverse* ou décalage) dans les cas où elle est non nulle.

Lorsque l'on compare la qualité des résultats entre un véritable calcul graphe *reverse* complet et le décalage des courses du trajet obtenu, on voit que cette dernière solution, bien que plus performante en temps d'exécution, n'est pas suffisante. En effet, seulement 20 % des trajets sont améliorés, en moyenne d'une heure et quart, comparé au 58 % d'amélioration pour le graphe *reverse* avec une moyenne d'une heure.

Méthodes comparées	Amélioration (%)	Temps gagné (min)
Décalage / Direct	14,8	63
Graphe *reverse* / Direct	58,2	77
Graphe *reverse* / Décalage	30,1	20

Tableau 19 – Qualité des solutions : décalage des courses et calcul du graphe *reverse*

En revanche, les temps de calcul du graphe *reverse* sont deux fois plus longs en moyenne, avec une dispersion importante, puisque cela dépend du nombre de solutions obtenues lors du calcul direct. Un grand nombre de calculs se fait en des temps trois ou quatre fois plus élevés que le calcul direct.
Ce calcul est néanmoins indispensable pour avoir un ensemble de solutions pertinent répondant à la requête initiale.

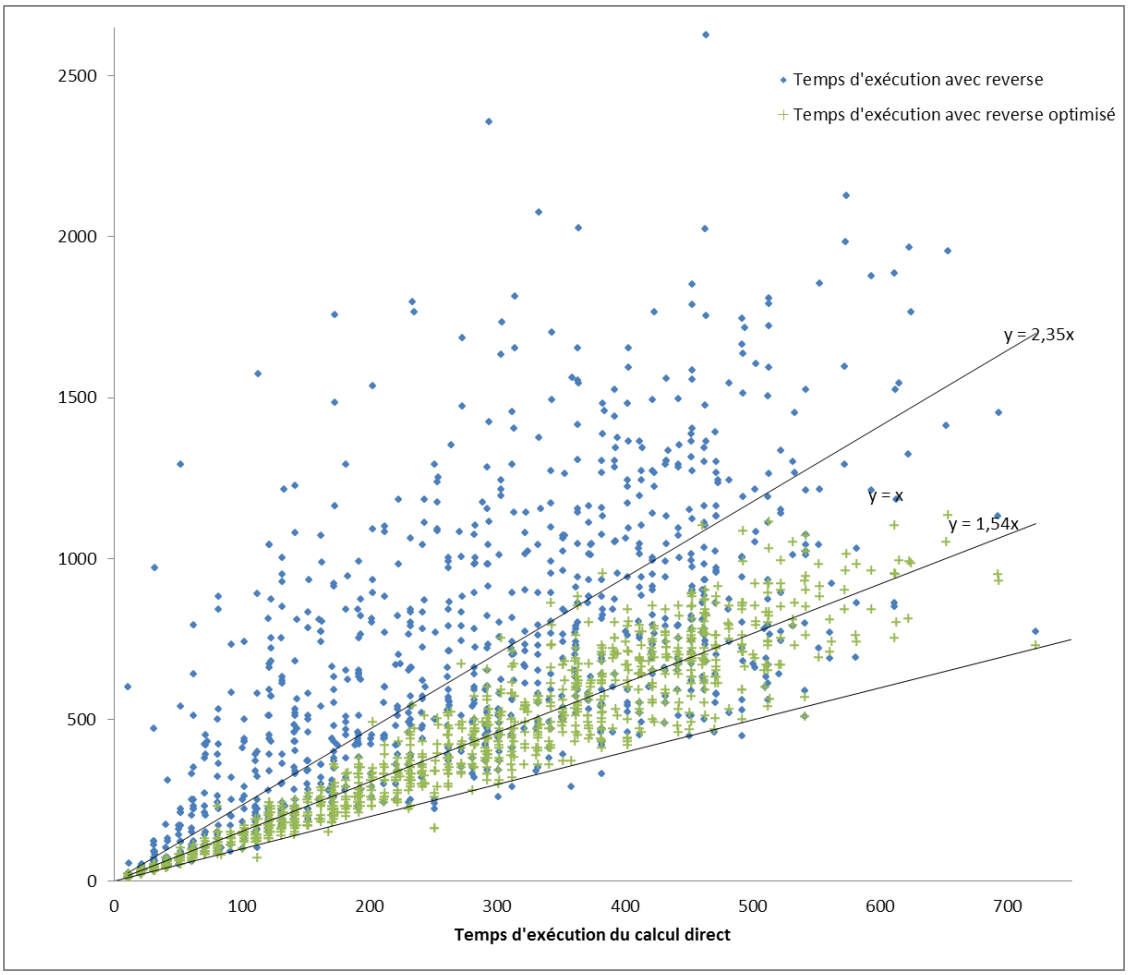


FIGURE 41 – Temps d'exécution du graphe *reverse* (R1)

Enfin, on a constaté que borner les calculs avec l'horaire de départ, vérifier la non dominance avec la dernière solution trouvée ou changer le tri du tas binaire n'ont pas eu d'impact significatif sur les temps de calcul.

En revanche, les optimisations par élimination de labels correspondant aux proprié-

tés 14, 15 et 16 améliorent les temps de calculs moyen d'environ 35 % et divisent le temps de calcul le plus long par plus de deux.

5 Conclusion

Nous avons vu dans ce chapitre comment une étude des ensembles de trajets et des critères de recherche a mené à l'établissement d'un algorithme exact de calcul d'itinéraire selon le critère **lexicographique de durée minimale**. Cet algorithme simple consiste à faire deux calculs : un premier calcul direct en *départ à*, et un deuxième calcul sur le graphe TC graphe *reverse* avec comme horaire d'arrivée l'horaire trouvé lors du premier calcul.

De plus, nous avons démontré et défini un autre algorithme exact pour le calcul d'itinéraire selon le bicritère **lexicographique, nombre de courses**. Cet algorithme reprend l'idée de refaire un calcul bicritère **horaire de départ, nombre de course** pour chaque horaire d'arrivée distinct trouvé lors d'un calcul bicritère **horaire d'arrivée, nombre de courses**. Les expérimentations montre que ces calculs résultent en une augmentation des temps de calcul moyens d'environ 150 %.

De nombreuses optimisations ont été développées afin de réduire ces temps CPU dans le cas bicritère, et les résultats expérimentaux finaux montrent une augmentation moyenne des temps de calcul de 50 % après optimisation.

La perspective de recherche sur ce sujet consiste principalement en l'étude et l'amélioration des performances de la recherche d'itinéraire multicritère prenant en compte le **critère lexicographique** et deux autres critères, voire plus.

Conclusion

Rappelons que la problématique générale de la thèse concerne la recherche de chemins multimodaux optimaux selon divers critères dans des réseaux de transport en commun et de transport individuel. Alors que la vision traditionnelle de la littérature dans ce domaine est généralement très théorique, nos travaux sont consacrés à l'évolution d'un **calculateur d'itinéraire industriel**. Utiliser un calculateur, ainsi que des données cartographiques et de transport, **issus du monde réel** est un ingrédient important dans l'obtention de résultats significatifs. Ce calculateur est le dernier maillon d'une chaîne d'applications qui permet à un utilisateur de site **Web** ou d'application **mobile** de planifier son trajet dans un espace géographique délimité.

Contributions

La première étape cruciale de nos recherches a été de construire un **état de l'art** aussi complet que possible, portant à la fois sur les **modélisations** et les **algorithmes** proposés dans la littérature. Parmi ceux-ci, nous nous sommes intéressés à la modélisation *time-dependant* du graphe de transport en commun, proche du modèle utilisé sur le calculateur existant. Nous avons aussi détaillé les différents paramètres, contraintes et critères de calcul que le calculateur prend déjà en compte, afin de montrer d'une part une **complexité** souvent ignorée dans la littérature, et d'autre part la sensibilité du code aux changements radicaux de modèle ou d'algorithme.

Nous avons choisi d'ignorer pour la suite de nos travaux les diverses techniques de la littérature **optimisant les temps CPU** des calculs d'itinéraire. En effet, dans un cadre d'utilisation Web ou mobile, les temps de calcul sont déjà largement inférieurs aux temps de préparation et transfert d'une réponse visuelle à l'utilisateur final. De plus, ces techniques demandent souvent des **prétraitements** de données assez lourds en temps CPU et en espace mémoire, en plus de dégrader la **flexibilité** du système.

Nous avons donc préféré explorer la piste de l'optimisation **multicritère**, absente du calculateur existant jusqu'alors (si ce n'est par le biais d'une somme pondérée), afin d'améliorer la **qualité** des résultats. Pour ce faire, nous avons transformé l'algorithme de Dijkstra utilisé par le calculateur en un **algorithme multilabel** capable de générer et conserver plusieurs trajets par nœud, selon des ressources et une règle de dominance génériques. Nous avons appliqué cet algorithme au cas bicritère (**arrivée au plus tôt**, **moins de changement**) et effectué des expérimentations sur plusieurs réseaux de transport en commun.

Une deuxième problématique que nous avons traitée est l'obtention de **trajets al-**

ternatifs susceptibles d'intéresser l'utilisateur final sans être nécessairement optimaux selon des critères bien définis. Nous avons résolu ce problème en le séparant en deux parties. D'abord, nous avons développé deux méthodes de **génération** de trajets non optimaux. La première est inspirée de l'algorithme des k-meilleures solutions et répète des calculs monolabels sous différentes contraintes. La deuxième est une variante de l'algorithme multilabel développé précédemment qui maintient une liste restreinte de labels par nœud. Ensuite, nous avons développé des méthodes de **sélection** des solutions, en se focalisant à la fois sur leur qualité (basée sur des critères objectifs) et leur diversité (par le biais d'une distance mesurant la différence entre deux solutions). Ces méthodes ont été testées et comparées entre elles sur un grand réseau de transport urbain et périurbain.

Enfin, nous avons considéré le problème du critère lexicographique d'**arrivée au plus tôt** puis **départ au plus tard** : étant donné un horaire de départ minimal, il ne suffit pas de trouver le trajet qui arrive le plus tôt possible, il faut également qu'il parte le plus tard possible. Cette question se résout facilement dans le cas de l'algorithme monolabel : il suffit de refaire un calcul en *arrivée à* qui optimise l'horaire de départ, avec l'horaire d'arrivée fixé par le premier calcul. En revanche, l'algorithme multilabel génère plusieurs trajets aux horaires d'arrivée différents. Par conséquent, optimiser le critère lexicographique et un autre critère (comme le nombre de changements) nécessite plusieurs calculs multilabels. Il nous a paru nécessaire de développer des optimisations permettant de maintenir un temps CPU global minimal. Nous avons défini formellement ce critère et démontré un certain nombre de propriétés justifiant les algorithmes développés et leurs optimisations. Enfin, les expérimentations effectuées génèrent des solutions bien meilleures du point de vue de ce critère, pour un temps de calcul augmenté de 50 % en moyenne.

Perspectives

La version multilabel du calculateur est flexible, au sens qu'elle permet l'implémentation de **nouvelles** ressources et règles de dominance spécifiques à tout ensemble de critères imaginables. Parmi les perspectives envisagées figurent donc la prise en compte de nouveaux critères, faisant ainsi profit de l'algorithme multilabel.

Un premier exemple d'évolution possible sur ce plan est le passage de l'algorithme bicritère lexicographique testé à un algorithme à trois critères minimisant en plus la **durée de marche à pied**. Nous pouvons alors anticiper que des optimisations seront nécessaires pour maintenir un temps CPU raisonnable du point de vue de l'utilisateur final.

Un autre sujet de travail riche en innovations potentielles est la prise en compte de la **tarification** des trajets en plus des critères usuels. C'est un sujet complexe car la tarification en transport en commun suit des règles parfois très spécifiques qui dépendent des opérateurs de transport. Le prix d'un ticket de transport peut être unitaire, payable à la montée, dépendre de l'origine et de la destination, ou dépendre d'une tarification zonale. Un périmètre géographique peut être desservi par plusieurs opérateurs : tous les types de tarification peuvent être utilisés pour un même calcul, ce à quoi se rajoutent souvent de nombreuses exceptions. Par exemple, les changements sont souvent gratuits

sur un même réseau pendant une certaine durée et/ou un nombre de changements maximum. De plus il existe des politiques de tarifications spéciales inter-opérateurs : par exemple, un ticket de métro peut être inclus dans le prix d'un trajet en car interurbain. Chaque type de tarif nécessite des développements spécifiques sur le calculateur multilabel. Ainsi, dans le cas d'une tarification origine/destination, le prix n'est connu qu'au moment de la descente, ce qui force à conserver des labels concurrents. L'ajout d'un critère du **moins cher** dans le cadre de l'algorithme multilabel est donc un vrai défi.

D'autres critères, comme le trajet le plus « **robuste** » (c'est-à-dire le trajet tel qu'un retard éventuel sur une des étapes ne rende pas les suivantes irréalisables), ou le trajet le plus « **confortable** » (avec des indices de confort provenant directement des utilisateurs des réseaux de transport) pourront également être envisagés.

La génération de trajets diversifiés du chapitre V est une première étape prometteuse qui ouvrent de nombreuses perspectives. Les méthodes monolabels peuvent voir leurs calculs **parallélisés** pour des meilleures performances globales, et la méthode multilabel peut être améliorée d'une part avec des **optimisations** heuristiques et d'autre part avec l'ajout d'une composante de calcul graphe *reverse* permettant d'ajouter les solutions partant au plus tard. De plus, les méthodes de sélection développées font un compromis entre qualité et diversité avec des algorithmes de regroupement, en choisissant les meilleures solutions de chaque groupe. D'autres méthodes d'aide à la **décision multicritère** originales ou issues de la littérature pourraient être développées et testées.

Une dernière piste d'amélioration concerne les **performances** du calculateur en temps de calcul CPU. Rappelons que nous avons recensé un certain nombre de méthodes d'accélération des calculs dans notre revue de la littérature. Vu que l'application de l'algorithme multicritère multilabel augmente les temps de calcul avec une intensité dépendant des critères choisis, il serait maintenant judicieux de vérifier quelles méthodes sont applicables à notre cas, et dans quelle mesure elles peuvent accélérer les calculs sans détériorer ni les temps de prétraitement ni la qualité des résultats.

Bibliographie

[Abra2011] I. Abraham, D. Delling, A. V. Goldberg, et R. F. Werneck *A hub-based labeling algorithm for shortest paths in road networks.* International Symposium on Experimental Algorithms, Springer, Berlin, Heidelberg, p. 230-241, 2011.

[Arti2013] C. Artigues, M.-J. Huguet, F. Gueye, F. Schettini, et L. Dezou *State-based accelerations and bidirectional search for bi-objective multi-modal shortest paths.* Transportation Research Part C: Emerging Technologies, vol. 27, p. 233-259, 2013.

[Ayed2011] H. Ayed *Approches de résolution multiobjective séquentielle et parallèle pour les réseaux de transports multimodaux* Thèse de doctorat, Université Paul Verlaine de Metz, 2011.

[Bast2006] H. Bast, S. Funke, et D. Matijevićs. *Transit: ultrafast shortest-path queries with linear-time preprocessing.* 9th DIMACS Implementation Challenge—Shortest Path, 2006.

[Bast2009] H. Bast *Car or public transport - two worlds.* Efficient Algorithms, vol. 5760, p. 355-367, 2009.

[BDGM2016] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor , P. Sanders, D. Wagner, R. F. Werneck. *Route planning in transportation networks.* Algorithm Engineering. Springer International Publishing, p. 19-80, 2016.

[BaDW2007] R. Bauer, D. Delling, et D. Wagner. *Experimental study on speed-Up techniques for timetable information systems.* Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS), p. 209-225, 2007

[Bell1956] R. Bellman *On a routing problem.* RAND Corporation Santa Monica, P-1000, 1956

[Bell1957] R. Bellman *Dynamic Programming* Princeton University Press, 1957

[Berg2009] A. Berger, D. Delling, A. Gebhardt, et M. Müller-Hannemann *Accelerating time-dependent multi-criteria timetable information is harder than expected.* 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS), 2009

[Bous2010] A. Bousquet *Optimisation d'itinéraires multimodaux fondée sur les temps de parcours à l'échelle d'une agglomération urbaine dense.* Thèse de doctorat, École Nationale des Travaux Publics de l'État, 2010.

[Brod2003] G. S. Brodal et R. Jacob *Time-dependent networks as models to achieve fast exact time-table queries.* Proceedings of ATMOS Workshop, p. 3-15, 2003.

[Defa1977] D. Defays *An efficient algorithm for a complete link method.* The Computer Journal, vol. 20, no 4, p. 364-366, 1977.

[Dell2009] D. Delling *Engineering and augmenting route planning algorithms.* Thèse de doctorat, Karlsruhe Institute of Technology, 2009.

[Dell2009a] D. Delling, T. Pajor, et D. Wagner *Engineering time-expanded graphs for faster timetable information.* Robust and Online Large-Scale Optimization, Springer Berlin Heidelberg, p. 182-206, 2009.

[Dell2012a] D. Delling, B. Katz, et T. Pajor. *Parallel computation of best connections in public transportation networks.* Journal of Experimental Algorithmics (JEA), vol. 17, p. 4.4, 2012

[Dell2012] D. Delling, T. Pajor, et R. F. Werneck. *Round-based public transit routing.* Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX), p. 130–140, 2012.

[Dell2013] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, et R. F. Werneck. *Computing multimodal journeys in practice.* International Symposium on Experimental Algorithms, Springer, Berlin, Heidelberg, p. 260-271, 2013

[Dibb2013] J. Dibbelt, T. Pajor, B. Strasser, et D. Wagner. *Intriguingly simple and fast transit routing.* International Symposium on Experimental Algorithms, Springer, Berlin, Heidelberg, p. 43-54, 2013.

[Dijk1959] Dijkstra, E. W. *A note on two problems in connexion with graphs.* Numerische mathematik, vol. 1, no 1, p. 269-271, 1959.

[Diss2008] Y. Disser, M. Müller–Hannemann, et M. Schnee. *Multi-criteria shortest paths in time-dependent train networks.* Experimental Algorithms, Springer, p. 347–361, 2008.

[Ehrg2006] M. Ehrgott *Multicriteria Optimization - Second Edition* Springer Science Business Media, 2006.

[Feki2010] F. Feki *Optimisation distribuée pour la recherche des itinéraires multi-opérateurs dans un réseau de transport co-modal* Thèse de doctorat, École Centrale de Lille, 2010.

[Geis2008] Geisberger, Robert *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.* Mémoire de master, Karlsruher Institut für Technologie, 2008.

[Geis2010] Geisberger, Robert *Advanced route planning in transportation networks.* Thèse de doctorat, Karlsruher Institut für Technologie, 2010.

[Gold2004] A.V. Goldberg, C. Harrelson *Computing the shortest path: A search meets graph theory.* Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, p. 156-165, 2005

[Gold2005] A.V. Goldberg, C. Harrelson, H. Kaplan, R. F. Werneck Présentation *Efficient Point-to-Point Shortest Path Algorithms.* http://www.cs.princeton.edu/courses/archive/spr06/cos423

[Gold2006] A.V. Goldberg, H. Kaplan, R. F. Werneck *Reach for A*: Efficient Point-to-Point Shortest Path Algorithms.* Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX), Society for Industrial and Applied Mathematics, p. 129-143, 2006.

[Grab2010] T.Gräbener *Calcul d'itinéraire multimodal et multiobjectif en milieu urbain.* Thèse de doctorat, Université de Toulouse, 2010.

[Gutm2004] R. Gutman *Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks.* Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments (ALENEX), p. 100-111, 2004.

[HaNi1968] P. E. Hart, N. J. Nilsson, et B. Raphael *A formal basis for the heuristic determination of minimum cost paths.* IEEE transactions on Systems Science and Cybernetics, vol. 4, no 2, p. 100-107, 1968.

[Hart1975] J. A. Hartigan *Clustering algorithms.* New York : Wiley, 1975.

[Kamo2007] A. Kamoun *Conception d'un système d'information pour l'aide au déplacement multimodal* Thèse de doctorat, École Centrale de Lille, 2007.

[Kirc2013] D. Kirchler *Efficient routing on multi-modal transportation networks.* Thèse de doctorat, École Polytechnique, Palaiseau, 2013.

[Laut2004] U. Lauther *An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background.* Geoinformation und Mobilität-von der Forschung zur praktischen Anwendung, vol. 22, p. 219–230, 2004.

[Liu2011] L. Liu *Data model and algorithms for multimodal route planning with transportation networks.* Thèse de doctorat, Technische Universität München, 2011.

[Loza2001] A. Lozano et G. Storchi *Shortest viable path algorithm in multi-modal networks.* Transportation Research Part A: Policy and Practice, vol. 35, no 3, p. 225-241, 2001.

[Macq1967] J. MacQueen *Some methods for classification and analysis of multivariate observations.* Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, no 14, p. 281-297, 1967.

[Mart1984] E. Q. V. Martins *On a multicriteria shortest path problem* European Journal of Operational Research, vol. 16, no 2, p. 236-245, 1984.

[Orda1991] A. Orda et R. Rom *Minimum weight paths in time-dependent networks.* Networks, vol. 21, no 3, p. 295–319, 1991.

[Pajo2009] Pajor, Thomas *Multi-Modal Route Planning.* Mémoire de master, Karlsruher Institut für Technologie, 2009.

[Pajo2013] Pajor, Thomas *Algorithm Engineering for Realistic Journey Planning in Transportation Networks.* Thèse de doctorat, Karlsruher Institut für Technologie, 2013.

[Pall1998] S. Pallottino et M. G. Scutella *Shortest path algorithms in transportation models: classical and innovative aspects.* Equilibrium and advanced transportation modelling, Springer US, p. 245-281, 1998.

[Pohl1969] I. Pohl *Bi-directional and heuristic search in path problems.* Thèse de doctorat, Stanford University, 1969.

[Pyrg2004] E. Pyrga, F. Schulz, D. Wagner et C. Zaroliagis *Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach.* Electronic Notes in Theoretical Computer Science, vol. 92, p. 85-103, 2004.

[Pyrg2008] E. Pyrga, F. Schulz, D. Wagner et C. Zaroliagis *Efficient models for timetable information in public transportation systems.* Journal of Experimental Algorithmics (JEA), vol. 12, p. 2.4, 2008

[Rama1997] M. V. Ramakrishna et J. Zobel *Performance in Practice of String Hashing Functions.* Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA), p. 215–224, 1997.

[Sand2006] P. Sanders et D. Schultes *Engineering Highway Hierarchies.* Algorithms – ESA 2006, p. 804-816, 2006.

[Sauv2011] G. Sauvanet *Recherche de chemins multiobjectifs pour la conception et la réalisation d'une centrale de mobilité destinée aux cyclistes.* Thèse de doctorat, Université Francois Rabelais - Tours, 2011.

[Schn2009] Schnee, Mathias *Fully realistic multi-criteria timetable information systems.* Thèse de doctorat, Technische Universität Darmstadt, 2009.

[ScWa1999] F. Schulz, D. Wagner et K. Weihe *Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport.* International Workshop on Algorithm Engineering, Springer, Berlin, Heidelberg, p. 110-123, 1999.

[Schu2005] F. Schulz *Timetable information and shortest paths.* Thèse de doctorat, Karlsruher Institut für Technologie, 2005.

[Sibs1973] R. Sibson *SLINK: An optimally efficient algorithm for the single-link cluster method.* The computer journal, vol. 16, no 1, p. 30-34, 1973.

[Stra2012] B. Strasser *Delay-Robust Stochastic Routing In Timetable Networks.* Thèse de doctorat, Karlsruher Institut für Technologie, 2012.

[Stra2014] B. Strasser et D. Wagner *Connection Scan Accelerated.* Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX), p. 125-137, 2014.

[Tung1992] C. T. Tung et K. L. Chew *A multicriteria Pareto-optimal path algorithm* European Journal of Operational Research, vol. 62, no 2, p. 203–209, 1992.

[Wang2004] J. Wang et T. Kaempke *Shortest route computation in distributed systems* Computers and Operations Research, vol. 31, no 10, p. 1621–1633, 2004.

[Wirt2015] A. Wirth *Algorithms for contraction hierarchies on public transit networks* Mémoire de master, Karlsruher Institut für Technologie, 2015.

[Yen1971] J. Y. Yen *Finding the k shortest loopless paths in a network* Management Science, vol. 17, no 11, p. 712-716, 1971

Glossaire

arc iti Dans le graphe TC *time-dependant*, décrit un arc décrivant une étape élémentaire sur un itinéraire, allant d'un nœud ITI à un autre nœud ITI. 62, 63, 86, 88, 89

arrêt physique Un arrêt physique, ou simplement arrêt, est un lieu géographique duquel il est possible de monter ou descendre de transport en commun. 13

correspwalk Dans le graphe TC *time-dependant*, décrit un arc de correspondance en marche à pied, allant d'un nœud MARCHE vers un nœud MARCHE. 62, 64, 66, 75, 84, 86, 88, 89, 106

course Une course est la description du parcours d'un véhicule de transport en commun. C'est en pratique une succession d'évènements (arrêt,horaire), avec les horaires croissants, accompagnée du mode de transport correspondant. 13

critère lexicographique Fait référence au critère **lexicographique de durée minimale**. Ce critère est un critère lexicographique toujours accompagné du critère d'**arrivée au plus tôt** ou du critère de **départ au plus tard**, et qui consiste à privilégier, parmi les solutions égales sur le critère principal, les solutions qui durent le moins longtemps.. 25, 63, 72, 116, 130

getin Dans le graphe TC *time-dependant*, décrit un arc de montée en transport en commun, allant d'un nœud MARCHE vers un nœud ITI. 62, 64, 84, 86, 88, 89, 93, 94, 106, 114, 115, 152

getoff Dans le graphe TC *time-dependant*, décrit un arc de descente de transport en commun, allant d'un nœud ITI vers un nœud MARCHE. 62, 64, 84, 86, 88, 89, 106, 115, 152, 153

graphe *reverse* Se dit d'un graphe TC dont tous les arcs sont inversés, et qui sert à calculer les trajets en *arrivée à*, que ce soit dans le cadre du critère de **départ au plus tard**, ou du critère **lexicographique**. Se dit également de l'algorithme (ou calcul) sur ce graphe.. 4, 62–64, 68, 72, 73, 110, 116, 130, 134, 135, 139, 144, 150, 151, 153, 156–158, 161, 169, 170

graphe tc Graphe représentant le réseau de transport en commun. 95, 158

horaire Un horaire est un moment de la journée, discrétisée en minutes ou secondes, utilisée pour indiquer le moment de passage d'un moyen de transport public à un arrêt. 13

itinéraire L'itinéraire d'une course est la succession des arrêts qu'elle parcourt. 14

ligne Une ligne de transport en commun est un regroupement de courses sous un même nom. 14

mode Un mode de déplacement correspond à une façon de se déplacer d'un endroit à l'autre. C'est un ensemble défini en extension, qui inclut, mais n'est pas limité à : *voiture, vélo, taxi, marche, train, car, bus, avion, métro, tram, covoiturage*. 11

nœud iti Dans le graphe TC *time-dependant*, c'est un nœud correspondant à un arrêt sur itinéraire.. 62, 63, 65, 66, 75, 84, 106, 114, 115, 152, 154, 155

nœud marche Dans le graphe TC *time-dependant*, c'est le nœud central d'un arrêt, autour duquel se font les changements vers des nœuds ITI et les correspondances vers d'autres nœuds MARCHE. 34, 62, 63, 67, 75, 90, 94, 115, 152, 154, 155, 168

RI Acronyme de recherche d'itinéraire. 23, 24, 27

TC Acronyme de transport en commun. 11, 15, 18–20, 23, 39, 54, 62, 64, 71–73, 95, 96, 144, 151, 153, 168

time-dependant Se dit d'un graphe TC dont les nœuds représentent des arrêts ou des arrêts sur itinéraire. L'extension d'un label sur un arc dépend d'un horaire au nœud de départ de l'arc. 2, 31, 33–35, 37, 38, 53–57, 59, 62, 65, 66, 76, 81–83, 111, 151, 159, 168

time-expanded Se dit d'un graphe TC dont les nœuds représentent des évènements. Les horaires et les courses sont intégrées dans les coûts des arcs, qui ne dépendent pas d'une ressource supplémentaire. 2, 34–38, 53–56, 58, 59, 64, 81, 82, 168

Table des figures

Liste des tableaux

Annexe A

Annexe : accrochage de lieux

Cette annexe est consacrée à une partie de nos travaux, en marge de l'évolution multilabel du calculateur d'itinéraire, consacrée à l'amélioration de la méthode permettant d'intégrer arrêts et lieux publics dans les graphes routiers de marche à pied, vélo et voiture.

1 Définition du problème

Rappelons que le **réseau de voirie** est constitué de tronçons, représentés par des lignes polygonales, et des arrêts et lieux publics, représentés par des coordonnées géographiques (dans une projection cartographique donnée).

Rappelons également que l'on modélise l'ensemble des tronçons de voirie par les arcs d'un graphe, reliant des nœuds représentant les extrémités de chaque tronçon.

Afin de relier les lieux géographiques avec le réseau de voirie, il faut un processus qui permet de les modéliser et de les connecter au reste du graphe.

Étant donné la quantité de tronçons (100k à 1M en fonction de la taille des réseaux), d'arrêts (5k à 50k) et de lieux publics (environ 2k à 20k) à considérer, ce processus ne peut pas être manuel, et doit pouvoir s'automatiser.

La solution adoptée est une recherche des plus proches voisins, et on considèrera ici que l'on sait résoudre ce problème de manière performante. Pour chaque couple de coordonnées (x, y) représentant un lieu à accrocher (on utilisera le terme anglais *snapping* plutôt que le mot accrochage), on trouve le tronçon le plus proche, puis on détermine le point de projection orthogonale du lieu sur le tronçon. Plus précisément, le tronçon étant composé de segments joints bout à bout, la projection orthogonale se fait sur la droite portée par le segment du tronçon le plus proche dudit point. On a alors deux cas, comme le montre la figure 42.

- Cas 1 : le point projeté fait partie du tronçon. On appelle l_1 la longueur de la partie du tronçon allant de son point de référence (*ref*) au point projeté et d la distance de projection.

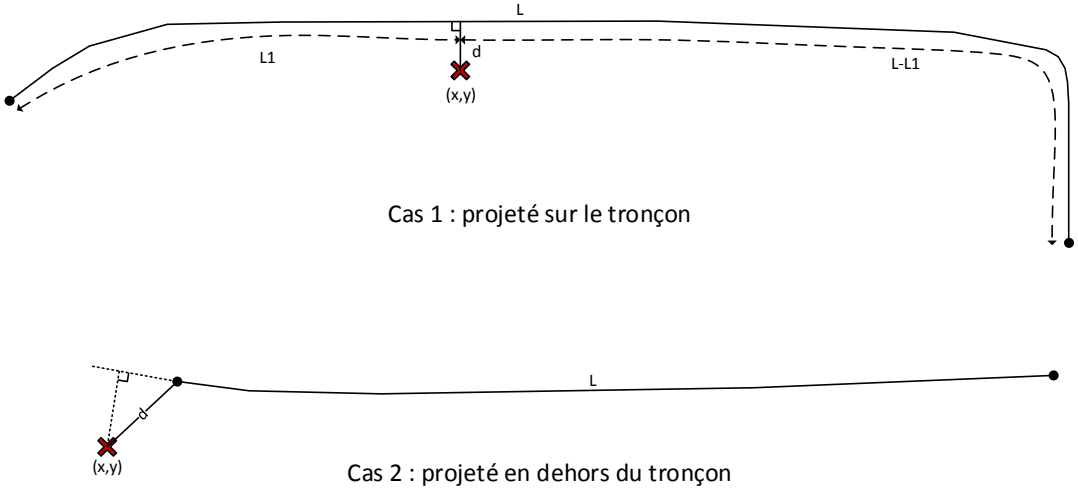


FIGURE 42 – Principe du *snapping*

- Cas 2 : le point projeté est aux limites ou en dehors du tronçon. On appelle d la distance de (x, y) au point le plus proche du tronçon, qui est forcément une des extrémités de ce dernier (*ref* ou *nref*). On définit $L_1 = 0$ ou bien $L_1 = L$ selon les cas.

On essaiera alors de modéliser dans le graphe le fait qu'il est possible de rejoindre le tronçon de longueur L en se déplaçant de d mètres, puis de rejoindre l'une ou l'autre des extrémités du tronçon en se déplaçant respectivement de L_1 ou $L - L_1$ mètre. C'est l'approche la plus naturelle, décrite en figure 43, mais d'autres approches sont possibles, comme on le verra plus tard.

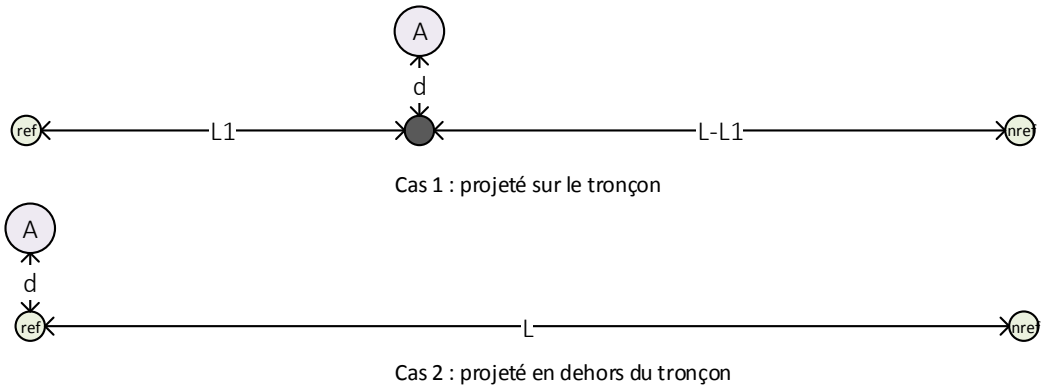


FIGURE 43 – Modélisation naturelle

1.1 Modélisation actuelle go@t

La modélisation actuelle utilisée par Cityway est très approximative : chaque arrêt est associé au tronçon le plus proche par la création de deux nœuds représentant l'arrêt. Ils sont reliés à ce tronçon par la création d'arcs de coût (en distance) $\mathtt{L}/2 + \mathtt{d}$, où $\mathtt{d}$ est la distance du lieu au point projeté.

Comme on peut l'imaginer, ce modèle peu précis génère un certain nombre de problèmes, en particulier pour les courts trajets à pied précédant ou suivant un mode de transport en commun.

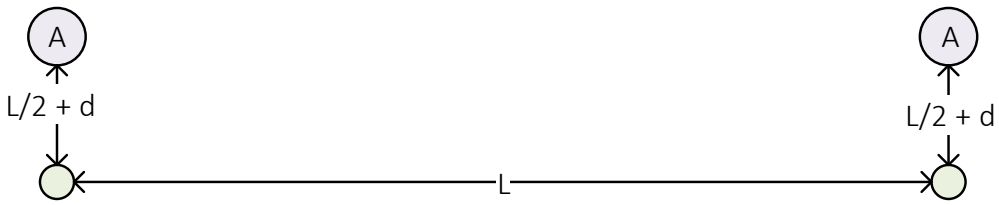


FIGURE 44 – Modélisation Go@t

1.2 *Snapping* forcé

Le fait de prendre le tronçon le plus proche quoi qu'il arrive peut entraîner des incohérences, comme de snapper à traver un immeuble, un mur, ou une autoroute (pour le mode marche à pied). Ce problème est résolu par une méthode de *snapping* manuel à un autre tronçon plus adéquat, qui annule le *snapping* automatique et force le lieu à être accroché à la bonne voirie. Les cas qui nécessitent cette approche ne sont pas suffisament nombreux pour justifier un procédé automatique.

La figure 45 est un exemple théorique de ce problème. Le lieu géographique représentant la gare est modélisé par un point dont le plus proche tronçon de voirie ne permet pas d'accéder à la gare. Cela fausse alors les calculs de distance et temps de marche à pied d'un lieu ou d'une rue avoisinant la gare vers celle-ci, détériorant en cascade tout calcul de plus court chemin multimodal partant ou arrivant dans les environs.

1.3 *multisnapping* 1-n

Un autre problème lié à cette méthode est l'existence potentielle de plusieurs accès d'une zone géographique, toujours représentée par un lieu géographique ponctuel, au réseau de voirie. Prenons l'exemple d'une gare à deux entrées, une à l'est donnant sur une voirie, l'autre à l'ouest donnant sur une autre voirie, dessiné sur la figure 46. Le *snapping* au tronçon le plus proche ne tient en conséquence pas compte de l'entrée 1 à l'ouest. Par conséquent, une recherche partant de la croix marquée *Lieu de départ* aura

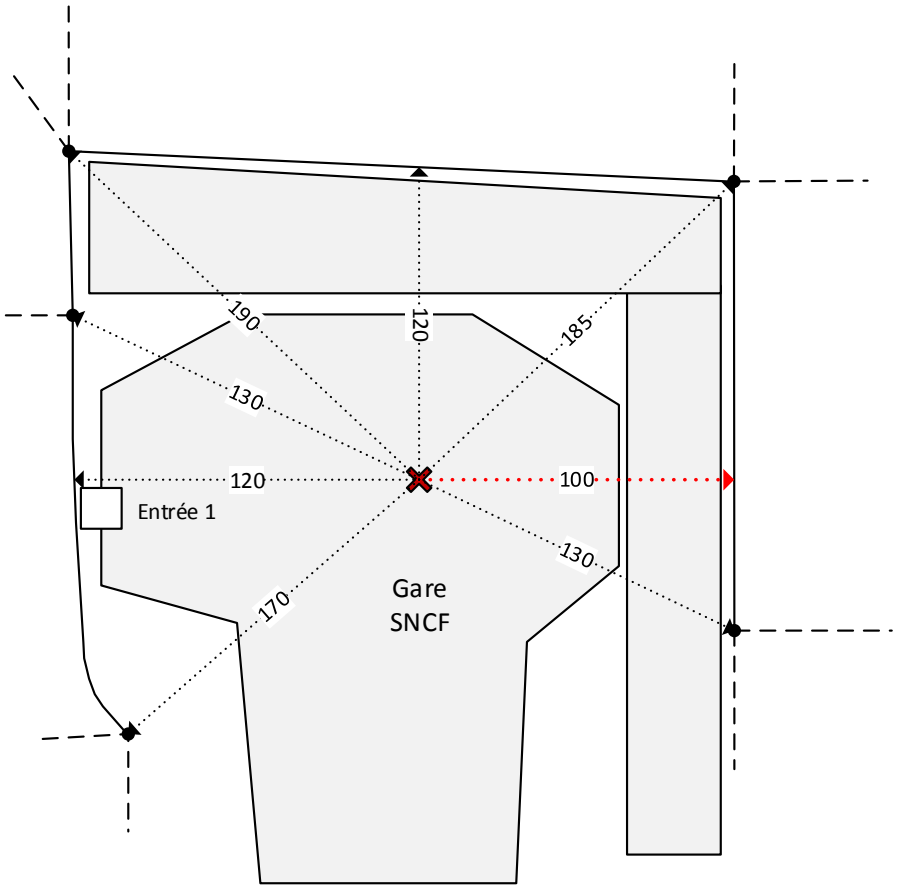


FIGURE 45 – Exemple d'un mauvais *snapping*

comme distance et temps de marche à pied de départ une valeur beaucoup trop grande, puisque le chemin le plus court reliant les deux croix passe par beaucoup plus de voiries que nécessaire en réalité.

On pourrait penser qu'il suffit de snapper automatiquement à un lieu géographique tous les tronçons dont la distance à ce lieu est inférieure ou égale à la distance minimale de *snapping* multipliée par un coefficient supérieur à 1. Cependant, l'exemple de la figure 45 ou l'exemple concret (voir 1.4) de la figure 47 montrent que cela ne fonctionne pas toujours. De plus, il n'y a finalement pas beaucoup de lieux pour lesquels le *multisnapping* se justifie, et l'on risque de faire artificiellement augmenter la taille du graphe en appliquant une forme automatique de *multisnapping*.

1.4 Exemple concret

Cet exemple représente la gare Part-Dieu à Lyon. On voit que la gare SNCF est représentée par un point au centre de la gare. Or il se trouve qu'un tronçon de voirie (colorié en vert sur la carte) accessible à pied et en voiture passe exactement sous la gare. Cependant d'autres entrées sont accessibles à pied et en voiture depuis les boulevards

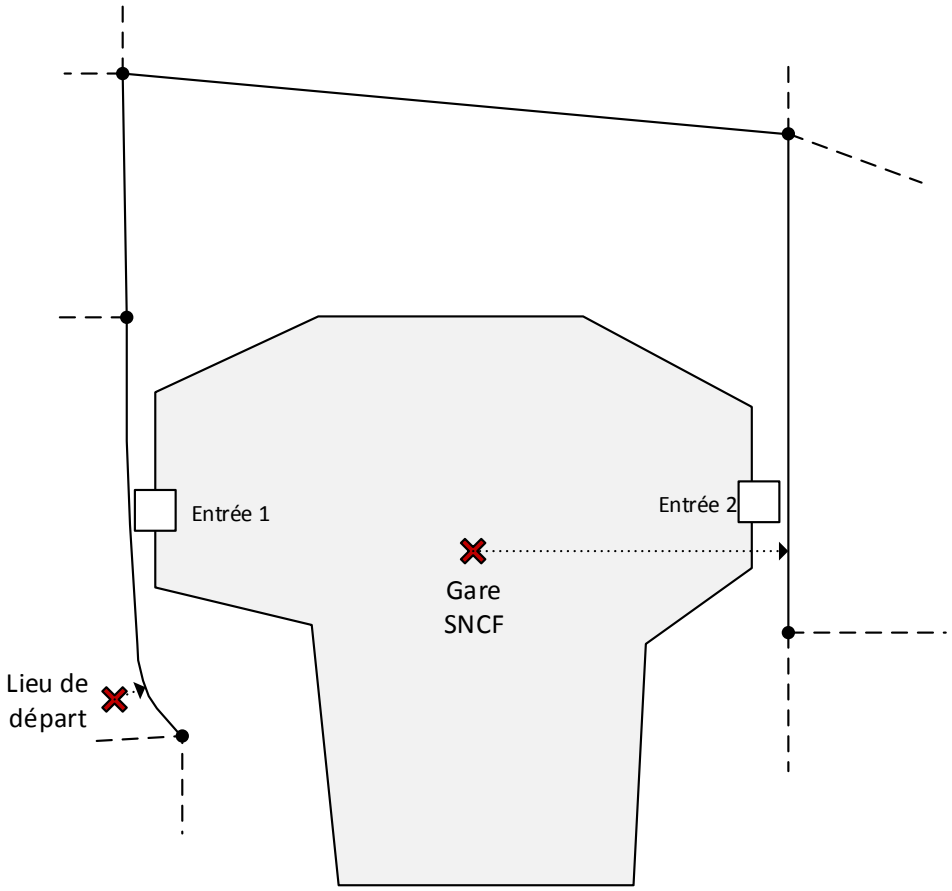


FIGURE 46 – Nécessité du *multisnapping*

extérieurs (coloriés en rouge sur la carte). Le système automatique actuel de Cityway ne nous permet pas de nous rendre à la gare sncf par une autre voirie que ce tunnel. Un grand détour est donc fait en rabattement voiture ou marche à pied vers la gare, alors que ça n'est pas nécessaire en réalité.

On constate également sur cette carte la présence de nombreux points d'arrêts (métro et bus) sur les voiries annexes, ce qui confirme la nécessité d'une bonne modélisation de plusieurs arrêts sur le même tronçon.

Enfin, on peut également observer que le *multisnapping* ne peut pas être automatisé en prenant les tronçons les plus proches : la distance de *snapping* nécessaire devrait être très élevée pour atteindre le tronçon rouge à l'est. Or il y a énormément de tronçons noirs (non accessibles directement depuis la gare), à des distances inférieures.

2 Modèles de *snapping*

Nous allons considérer plusieurs modèles alternatifs au modèle actuel utilisé, tous décrits dans la figure 48.

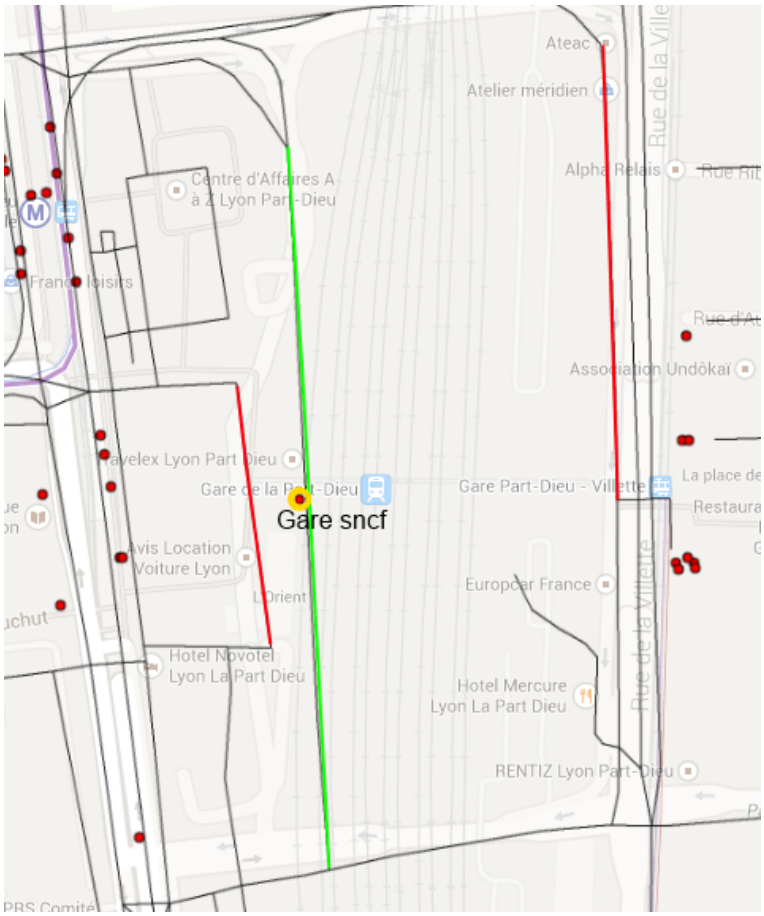


FIGURE 47 – Exemple réel : gare de la Part-Dieu

Les éléments ci-dessous devront être pris en compte lorsque l'on détaillera les modèles.

Précision Plus le modèle est proche de la réalité, moins les calculs de recherche d'itinéraire seront faussés.

***multisnapping* 1-n** Facilité de *snapping* d'un même arrêt à plusieurs tronçons.

***multisnapping* n-1** Facilité de *snapping* de plusieurs arrêts au même tronçon.

Vitesse modifiée Les informations historisées, en temps réel ou les perturbations modifient la vitesse moyenne en voiture de certains tronçons. Il faut s'assurer dans le modèle qu'il reste possible et aisé de modifier la vitesse d'un tronçon cartographique dans un sens donné.

Départ avec distance de décalage Facilité d'implémenter une RI dont un point de départ ou d'arrivée est précisément défini. C'est-à-dire plutôt que de partir d'un tronçon (c'est-à-dire des nœuds ref et nref du tronçon), partir d'une position géographique sur le tronçon, définie par l'identification du tronçon, le sens de parcours et une distance au nœud ref.

Graphe multicouche Facilité d'implémenter un graphe multicouche, c'est-à-dire de connecter les nœuds représentant les arrêts au graphe TC.

Nombre d'arcs rajoutés Le temps de calcul total est fortement lié au nombre d'arcs du graphe.

2.1 Calculateur de Cityway

Ce modèle est le modèle actuel utilisé par Cityway. Chaque lieu à snapper génère deux nœuds, puis chaque duplicat est relié l'un à ref, l'autre à nref par plusieurs arcs ayant les mêmes propriétés que les arcs représentant le tronçon (catégorie de vitesse). Les distances des nouveaux arcs sont égales à la moitié de la longueur du tronçon snappé.

Précision Très mauvais.

***multisnapping* 1-n** Duplication de plusieurs nœuds.

***multisnapping* n-1** Mauvais : impossible d'aller d'un lieu à l'autre sans passer par l'extrémité d'un tronçon même s'ils sont à 1 mètre l'un de l'autre.

Vitesse modifiée Répercussion sur tous les arcs dupliqués en faisant attention au sens.

Départ avec distance de décalage Mauvais : point de départ réparti sur tous les duplicats d'arcs.

Graphe multicouche Compliqué : plusieurs nœuds représentant le même arrêt.

Nombre d'arcs rajoutés Linéaire élevé. Pour n arrêts : $2n$ nœuds et $4n$ arcs.

2.2 Réaliste

Un modèle intuitif qui consiste pour chaque lieu connecté à un tronçon à séparer l'arc représentant le tronçon en deux arcs partageant les mêmes propriétés en créant un nœud de projection intermédiaire et un nœud unique représentant le lieu a snapper, relié à ce nœud de projection par des arcs d'un nouveau type (projection) – voir les figures 43 et 48.

Précision Très bon. Le seul problème est que la distance réelle à parcourir du lieu vers la voirie pourrait être différente de la distance de projection.

***multisnapping* 1-n** Bon : le même nœud peut être connecté à des nœuds d'intersection sur d'autres tronçons.

***multisnapping* n-1** Bon : il est possible d'aller d'un nœud à l'autre en passant par une fraction du tronçon et les arcs projetés.

Vitesse modifiée Répercussion sur tous les arcs représentant le tronçon, pas sur les arcs de projection.

Départ avec distance de décalage Excellent : point de départ unique clairement défini sur l'arc représentant le tronçon.

Graphe multicouche Bon : un seul nœud par arrêt, potentiellement connecté à d'autres graphes.

Nombre d'arcs rajoutés Linéaire élevé. Pour n arrêts : $2n$ nœuds et $4n$ arcs.

2.3 Réaliste simplifié

Pour réduire le nombre d'arcs, on peut simplifier le modèle en approximant la distance de projection à 0. Alors, les nœuds de projection sont confondus avec les nœuds représentant les lieux à snapper. Ce modèle est dessiné dans la figure 48 b).

Précision Moyenne. Si le lieu est éloigné de la voirie de plusieurs centaines de mètres, cela peut fausser significativement la distance réelle de parcours.

***multisnapping* 1-n** Mauvais : on est obligé de dupliquer les nœuds représentant le même arrêt sur plusieurs tronçons différents.

***multisnapping* n-1** Bon : il est possible d'aller d'un nœud à l'autre en passant par une fraction du tronçon.

Vitesse modifiée Répercussion sur tous les arcs représentant le tronçon.

Départ avec distance de décalage Excellent : point de départ unique clairement défini sur l'arc représentant le tronçon.

Graphe multicouche Mauvais : plusieurs nœuds par arrêt, potentiellement connectés à d'autres graphes.

Nombre d'arcs rajoutés Linéaire minimal. Pour n arrêts : n nœuds et $2n$ arcs.

2.4 Bousquet [Bous2010]

L'idée est de rajouter un nœud par lieu à snapper à un tronçon de voirie, puis de les relier aux nœuds ref et nref. Ensuite il faut relier tous les nœuds représentant des lieux snappés au même tronçon entre eux en utilisant la distance réelle entre les deux (issues du modèle réaliste par exemple).

Précision Bonne. Identique au modèle réaliste.

***multisnapping* 1-n** Bon : pas la peine de dupliquer les nœuds, on peut créer les arcs reliant un nœud à un autre tronçon de la même façon. Cela démultiplie le nombre d'arcs de manière considérable puisqu'il faut aussi le relier à tous les nœuds représentant les lieux snappés à l'autre tronçon.

***multisnapping* n-1** Bon : il est possible d'aller d'un nœud directement à un autre.

Vitesse modifiée Compliqué : il faut ordonner les lieux en fonction de leur distance au point de référence pour appliquer les modifications de vitesse dans le bon sens sur les arcs directs.

Départ avec distance de décalage Mauvais : point de départ sur plusieurs arcs.

Graphe multicouche Bon : un seul nœud par arrêt.

Nombre d'arcs rajoutés Linéaire en nombre de nœuds et quadratique en nombre d'arcs rajoutés. Pour n arrêts : n nœuds et $n^2 + 3n$ arcs.

2.5 Discussion

La méthode actuelle de *snapping* est largement améliorable. Nous allons le faire de deux façons.

D'abord, nous allons adapter la modélisation des données au **modèle réaliste**, ce qui permettra d'avoir une qualité de résultat irréprochable, et permettra de faire des requêtes avec un point de départ précis sur un tronçon. Ensuite, nous adapterons la génération du graphe avec la possibilité de snapper certains lieux problématiques à de multiples tronçons de voirie, à partir de données manuelles.

3 Résolution

Tout d'abord nous allons voir quelques notions de géométrie euclidienne, puis nous détaillerons les algorithmes de construction de graphe snappé.

3.1 Distance

L'espace est doté d'une distance euclidienne entre deux points A et B, notée $d(A, B)$, étendue à tout couple de sous-ensemble de l'espace comme la borne inférieure de toutes les distances entre chaque point de ces sous-ensembles.

On présuppose ici que l'on sait calculer cette distance et résoudre efficacement le problème de recherche des plus proches voisins. C'est-à-dire qu'étant donné un grand ensemble de tronçons de voirie éligibles inclus dans $\mathcal{G}$ et un lieu géographique L, on sait trouver le tronçon $T \in \mathcal{G}$ tel que la distance entre T et L soit minimale avec une complexité temporelle faible.

On notera $T = (t_1, ..., t_n)$, $t_i = (S_i, S_{i+1})$ les segments orientés définissant le tronçon.

On notera $l(t_i) = d(S_i, S_{i+1})$ la longueur de chaque segment.

Notons l la longueur totale du tronçon.

3.2 Point le plus proche

Définition 45 *On appelle* ***point le plus proche*** *d'un point* P *dans un sous-ensemble de l'espace* G *le point de* G *qui minimise sa distance avec* P.

Pour trouver le point le plus proche d'un lieu L dans un segment $t = [S_1, S_2]$, on calcule à la fois la distance et le point le plus proche par la méthode suivante.

- Calcul du point projeté P et de la distance $d(P, L)$ de L sur la droite portée par t avec un produit scalaire.
- Si P est sur t, le point le plus proche est P et la distance $d(L, t) = d(P, L)$.
- Si P n'est pas sur t, le point le plus proche est S_1 ou S_2 et $d(L, t) = \min(d(S_1, L), d(S_2, L))$.

Étendre ce calcul pour trouver le point le plus proche et la distance d'un point L à un tronçon $T = (t_1, ..., t_n)$ est trivial. Il suffit de choisir un segment t_k qui minimise $d(L, t_i)_{i\in[1;n]}$: c'est également la distance de L à T et le point le plus proche de ce segment est aussi le point le plus proche de T à L.

3.3 Abscisse de coupure

Définition 46 *Étant donné un tronçon de voirie* T*, un lieu géographique* L *et notons* $P \in T$ *le point le plus proche de* L *dans* T*. On appelle* ***abscisse de coupure*** *de* L *sur* T *la longueur du sous-tronçon de* T *formé par l'ensemble des segments de* T *allant du début du tronçon au point* P.

Plus explicitement, soit k l'indice d'un segment t_k le plus proche de L.

- **Si** P **est sur** t_k, le point le plus proche de L_j sur le segment est P. On calcule l'abscisse de coupure l du point projeté P sur le tronçon T avec la formule suivante, où $l(t)$ signifie longueur du segment t.

  $l = \sum_{i=1}^{k-1} l(t_i) + d(S_i, P)$.
- **Si** P **n'est pas sur** t_k, le point le plus proche de L sur le segment est S_j avec $j = k$ ou $j = k + 1$. On a :

  $l = \sum_{i=1}^{j} l(t_i)$.

3.4 Algorithme détaillé de découpage des tronçons et altération du graphe

Pour chaque tronçon de voirie T, on suppose connu les lieux géographiques $\mathcal{L}(T)$ que l'on doit snapper à T.

3.5 Notations

Voici les notations et fonctions utilisées dans la description de l'algorithme.

- newNode() fait référence à une procédure de création et d'ajout d'un nœud au graphe existant.
- connect(N1, N2) fait référence à une procédure de création et d'ajout d'arcs reliant les nœuds N1 et N2 au graphe existant.
- deleteOldLinks(T) fait référence à une procédure qui supprime du graphe existant les arcs représentant le tronçon T
- $\mathsf{l}(\mathsf{L}, \mathsf{T})$ est l'abscisse de coupure de L sur T.
- $\mathsf{d}(\mathsf{L}, \mathsf{T})$ est la distance de L à T.
- ref(L) est le nœud de référence de L (correspondant à S_1).
- nref(L) est le nœud de fin de L (correspondant à S_n).
- cutLengthsMultiMap est une structure de donnée de type hashmap clé/valeur avec plusieurs valeurs possibles pour la même clé.
- cutLengths est une liste.

3.6 Algorithme

L'algorithme 13 définit comment passer d'un graphe représentant le réseau routier sans aucun lieu ou arrêt à un graphe représentant le même réseau, relié à des nœuds représentant les arrêts et les lieux géographiques à snapper. Certains détails ont été omis par souci de clarté, en particulier en ce qui concerne l'affectation des attributs des nouveaux arcs créés, comme le coefficient de sécurité en vélo, la vitesse moyenne de parcours dans divers modes, etc.

3.7 Exemple

Un exemple est décrit en figures 49 et 50 appliqué à un tronçon à 3 segments avec 4 lieux.

3.8 Approximations et simplification

Pour limiter de manière aisée le nombre de nœuds et d'arcs à rajouter au graphe, on pourra éviter la création d'un nœud de coupure entre ref et nref si la distance de coupure est suffisamment proche de 0 ou suffisamment proche de l (on pourra tester plusieurs seuils de fusion des nœuds comme 1,2, 5 et 10 mètres). De la même manière on pourrait penser à fusionner les nœuds de coupure entre eux s'ils sont reliés par un arc de coût inférieur à ce seuil. Pour conserver la longueur totale du tronçon, et la cohérence des résultats, il faut déporter l'approximation sur l'arc suivant les nœuds fusionnés, comme détaillé dans l'algorithme 14.

Enfin une deuxième optimisation pourrait être de fusionner les nœuds de lieux avec leurs nœuds de coupure respectifs, si celui-ci est unique (monosnapping), si le coût de l'arc reliant ces nœuds est inférieur à un seuil, et si le nœud de coupure n'est pas déjà fusionné avec un autre nœud de coupure.

Algorithme de fusion des cutnodes L'idée de l'algorithme est de créer une agrégation des longueurs de coupure en les rapprochant en priorité de 0 ou de la longueur totale du tronçon (pour éviter de créer des nœuds potentiellement inutiles) puis de regrouper les longueurs par paquets dont les valeurs ne peuvent pas être distantes de plus d'un seuil paramétrable. On prendra comme valeur de longueur de coupure d'un paquet la médiane des valeurs et on associera tous les lieux liés à chaque valeur d'une longueur d'un paquet à la médiane du paquet.

- mergeThreshold est un entier représentant le seuil de fusion des nœuds que l'on souhaite appliquer.
- L'algorithme de fusion s'insère dans l'algorithme précédent en prenant en paramètre la structure cutLengthsMultiMap
- On a une nouvelle structure du même type newCutLengthsMultiMap.
- aggregatedCutLengthsList est une liste de listes d'entiers permettant le regroupement des valeurs.
- $\mathfrak{l}(\mathrm{L})$ est la longueur totale du tronçon que l'on considère.

La figure 51 montre le modèle de nœuds de coupure fusionnés entre eux ou avec les nœuds extrêmes, avec un seuil de fusion de 5. La figure 52 montre la fusion entre les nœuds de lieux et les nœuds de coupure.

Effet sur la taille des graphes Les tables 3.8 et 3.8 décrivent la taille d'un graphe routier représentant la région PACA entière, sur lequel viennent se greffer environ 30000 arrêts et 8600 points d'intérêt. La table 3.8 compare les tailles du graphe simple, sans lieux incorporés, avec l'implémentation initiale de Cityway de la section 2.1. La table 3.8 montre les tailles des graphes avec la méthode réaliste décrite dans cette section. Vu que le nombre d'arrêts et lieux est assez faible par rapport au nombre de tronçons de voirie, la quantité de nœuds et arcs économisées par la méthode de fusion est assez faible : environ 2 % avec une perte de précision de 10 mètres. Dans l'implémentation, on restera sur un seuil de fusion nul, c'est-à-dire sur la version naturelle la plus proche de la réalité. Si les calculs sont trop longs, on cherchera à optimiser par diverses heuristiques classiques de recherche d'itinéraire (A*, ALT...).

	Nombre d'arcs	Nombre de nœuds
Graphe sans lieux	1024077	478129
Snapping Cityway	1158497	551889

Tableau 20 – Taille des graphes

Seuil	Nb arcs	Nb nœuds	Gain arcs	Gain nœuds
0	1157142	547548	0	0
2	1151767	544636	5375	2912
5	1144977	540925	12165	6623
10	1134982	535400	22160	12148

Tableau 21 – Effet de la fusion des nœuds sur la taille des graphes

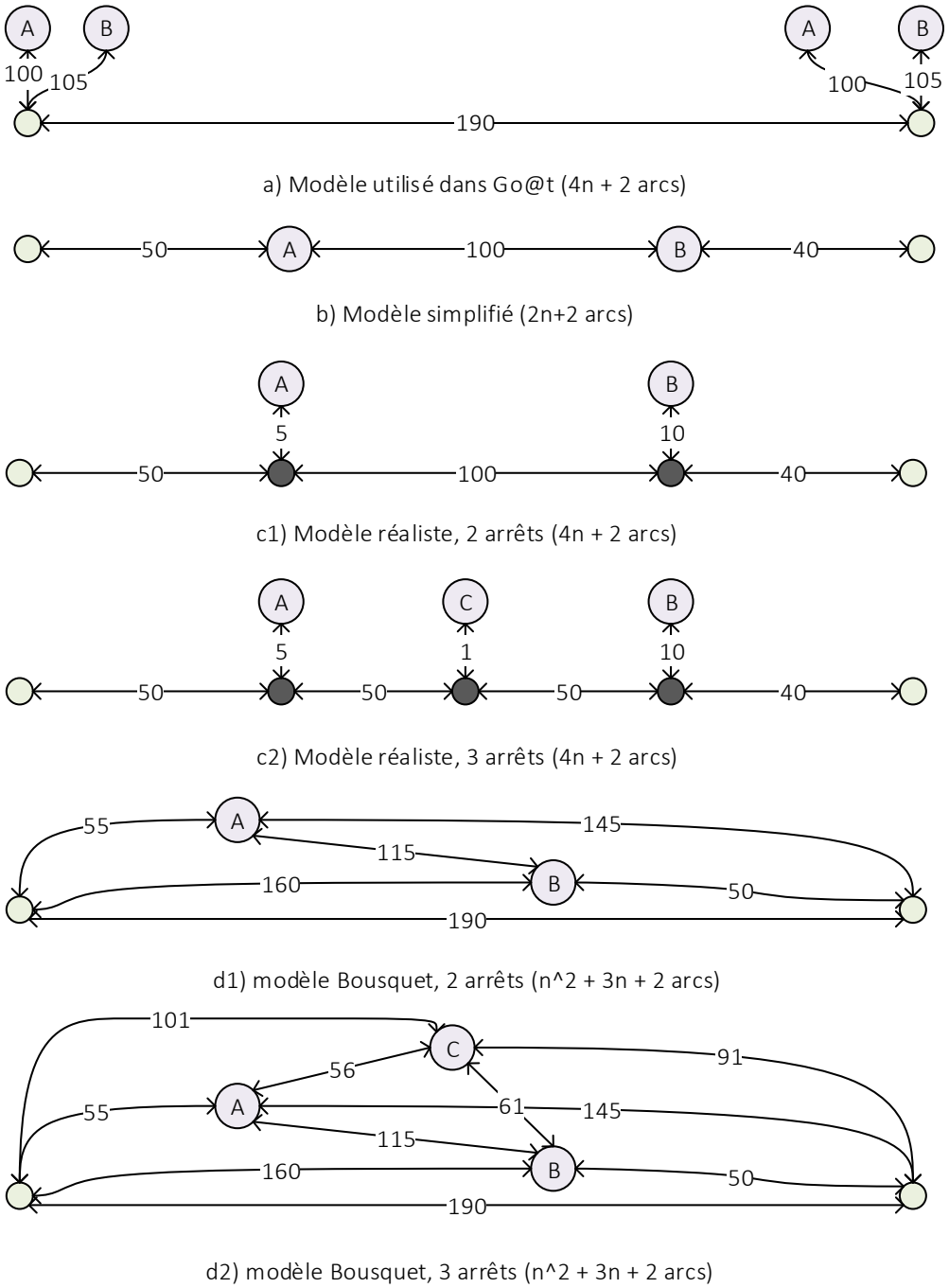


FIGURE 48 – Différents modèles de 2 ou 3 arrêts snappés au même tronçon

Algorithm 13 Algorithme de *snapping* réaliste

```
{Création des nœuds de location}
for all L do
  N(L) ← newNode(L)
end for
{Création des nœuds de coupure et des arcs}
for all T do
  for all L ∈ ℒ(T) do
    cutLengthsMultiMap[l(L, T)].Add(L)
  end for
  cutLengths = cutLengthsMultiMap.Keys()
  Sort(cutLengths)
  prevNode ← ref(L)
  prevLength ← 0
  hasCutNodes ← false
  for all l ∈ cutLengths do
    linkLength ← l− prevLength
    if l = l(L) then
      currNode ← nref(L)
    else if l = 0 then
      currNode ← ref(L)
    else
      currNode ← newNode()
      hasCutNodes ← true
    end if
    for all L ∈ cutLengthsMultiMap[l] do
      Connect(N(L), currNode)
    end for
    if hasCutNodes and currNode ≠ prevNode then
      Connect(currNode, prevNode)
    end if
    prevNode ← currNode
    prevLength ← l
  end for
  if hasCutNodes and prevNode ≠ nref(T) then
    Connect(nref(T), prevNode)
  end if
  if hasCutNodes then
    DeleteOldLinks
  end if
end for
```

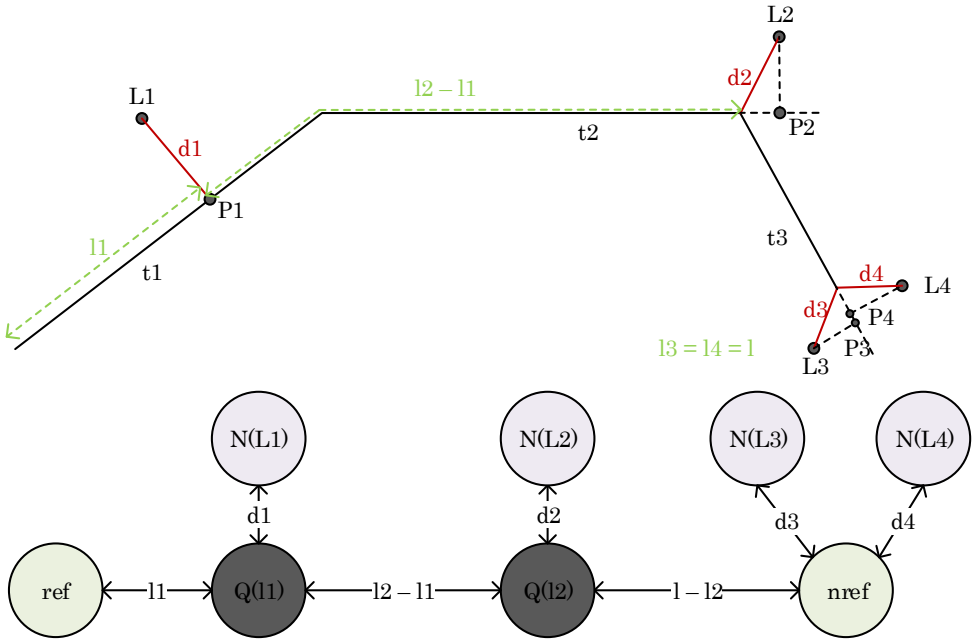


FIGURE 49 – Exemple de l'algorithme de *snapping*

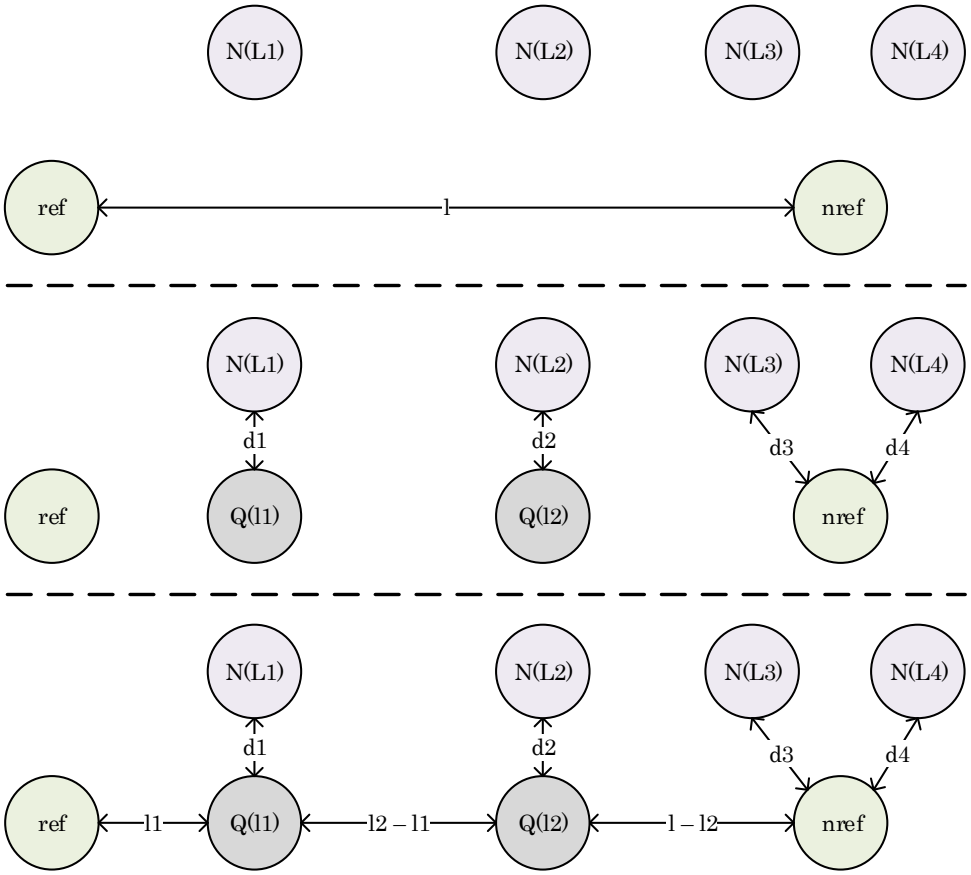


FIGURE 50 – Exemple : évolution du graphe lors de l'exécution de l'algorithme de *snapping*

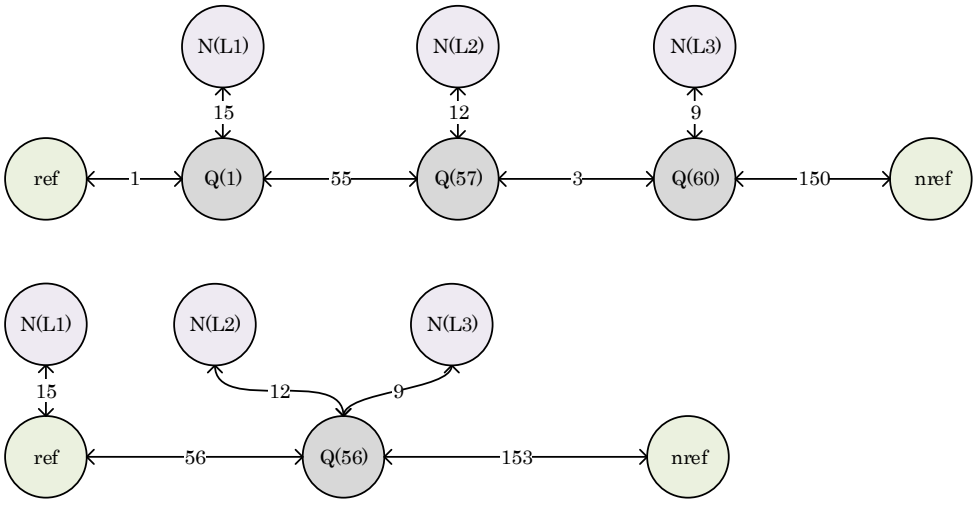


FIGURE 51 – Fusion des nœuds de coupure entre eux

Algorithm 14 Fusion des cutnodes

```
Initialize multimap newCutLengthsMultiMap
Initialize list refCutLengthsList
Initialize list nrefCutLengthsList
Initialize list aggregatedCutLengthsList
firstCutLength ← 0
cutLengths ← cutLengthsMultiMap.Keys().Sort()
for all l ∈ cutLengths do
  if l ≥ l(L)− mergeThreshold and l(L) − l < l then
    nrefCutLengthsList.Add(l)
  else if l ≤ mergeThreshold then
    refCutLengthsList.Add(l)
  else if firstCutLength > 0 and l - firstCutLength ≤ mergeThreshold then
    aggregatedCutLengthsList.Last().Add(cutLength)
  else
    firstCutLength ← l
    aggregatedCutLengthsList.Add(new List)
    aggregatedCutLengthsList.Last().Add(l)
  end if
end for
for all l ∈ refCutLengthsList do
  pointsToSnapByAggregatedCutLength.[0] ← cutLengthsMultiMap[l]
end for
for all l ∈ nrefCutLengthsList do
  pointsToSnapByAggregatedCutLength.[l(T)] ← cutLengthsMultiMap[l]
end for
for all list∈ aggregatedCutLengthsList do
  approx ← (Max(list) + Min(list)) / 2;
  for all l ∈ list do
    pointsToSnapByAggregatedCutLength.[approx] ← cutLengthsMultiMap[l]
  end for
end for
```

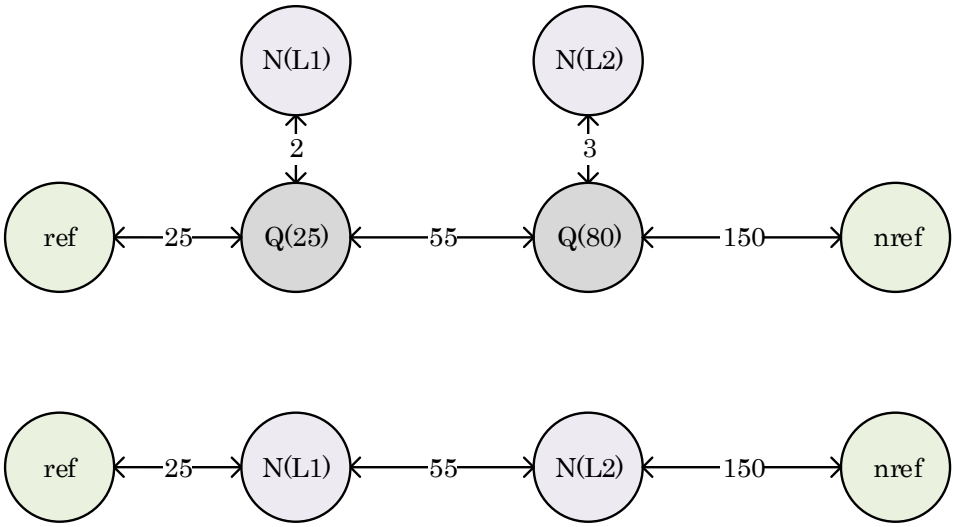


FIGURE 52 – Fusion des nœuds de lieux avec leurs nœuds de coupure

3.9 Autour des gares

Nous allons voir un certain nombre de cartes de gares, sur lesquelles sont dessinées les entrées en bleu. Un hypothétique algorithme de *snapping* devrait évaluer les points d'accès les plus probables pour chaque cas. Mais on voit sur les figure 53, 56, 54 et 55 que chaque gare est unique, et qu'il n'y a pas de règle qui régit la disposition des entrées des gares.

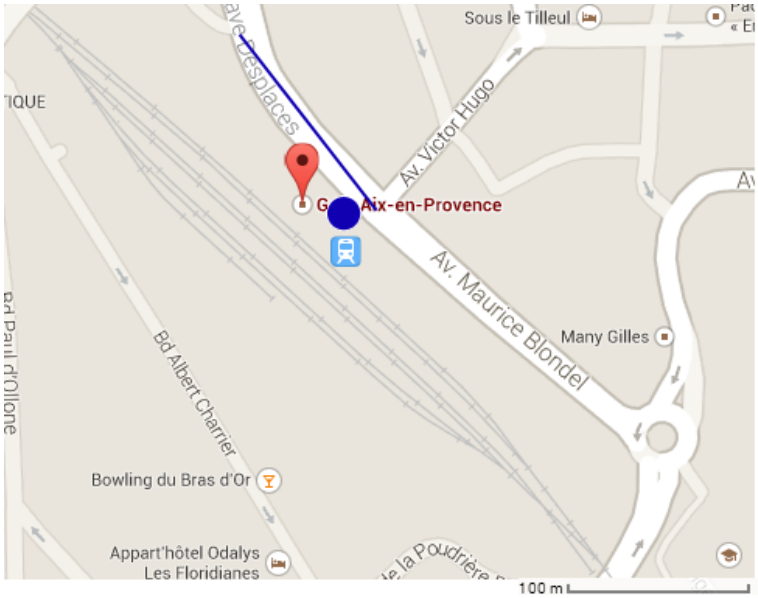


FIGURE 53 – Gare d'aix centre : une seule entrée mais deux routes entourant la gare

L'idéal est par conséquent d'obtenir par un moyen ou un autre les localisations des différents points d'accès aux gares :

- Entrées piétonnes, temps d'accès moyen à la gare (ascenseurs, tapis mécaniques),
- Parking et temps de marche jusqu'aux quais,
- Dépose minute.

Dans le cas optimiste où l'on dispose de ces informations, on pourrait définir un algorithme de *multisnapping* avec ces informations qui respecte à la fois les temps d'accès (marche, ascenseur) et les modes accessibles (voiture/piéton). On commencerait par snapper les entrées au graphe en associant des coûts adéquats par la méthode décrite dans ce document, puis on supprimerait les nœuds représentant les entrées pour relier le nœud représentant la gare directement aux nœuds de coupure en additionnant les distances. Ainsi la figure 57 représenterait le graphe correspondant à la gare d'Aix TGV décrite par la figure 56.

4 Conclusion

Nous avons donc présenté brièvement dans cette annexe un problème fortement lié à la recherche d'itinéraire : le rattachement des lieux publics et arrêts physiques aux

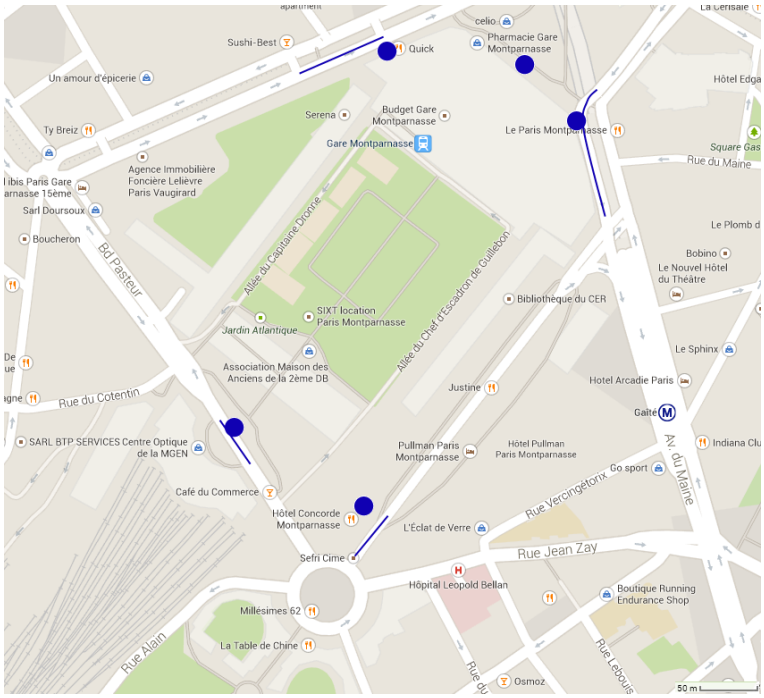


FIGURE 54 – Gare de Montparnasse : plusieurs bâtiments, 4 niveaux, des ascenseurs, parkings et dépose minute

réseaux routiers individuels. Une première contribution a été de changer le modèle de Cityway pour un modèle à la fois plus intuitif, plus précis et limitant le nombre d'arcs et de nœuds rajoutés. Une deuxième contribution a été de développer les outils effectuant les algorithmes de *snapping*. Enfin nous avons testé des algorithmes de fusion des nœuds proches pour limiter la taille du graphe, mais nous avons choisi de ne pas donner suite à cette idée, le nombre de nœuds rajoutés par le *snapping* étant trop faible comparé au nombre de nœuds du graphe pour induire une différence en terme de temps de calcul CPU. Enfin nous avons montré l'utilité du *multisnapping* et présenté les difficultés liées à l'automatisation du processus. Nous avons présenté un modèle pouvant résoudre ce problème dans le cas où les entrées des gares et autres lieux à entrées multiples sont fournies, mais n'avons pas développé ni testé ce modèle.

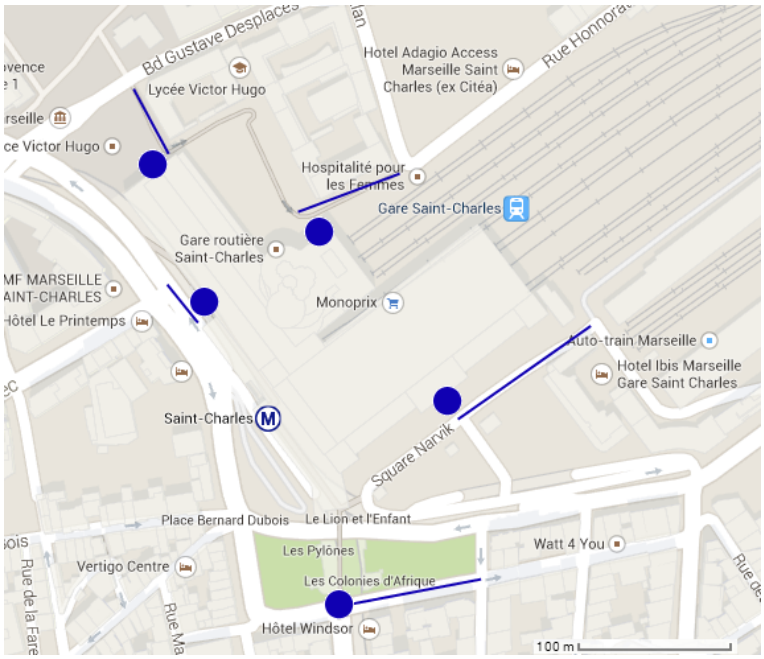


FIGURE 55 – Gare Saint-Charles : escaliers d'accès par dessus un minitunnel, 4 entrées, pas de dépose minute

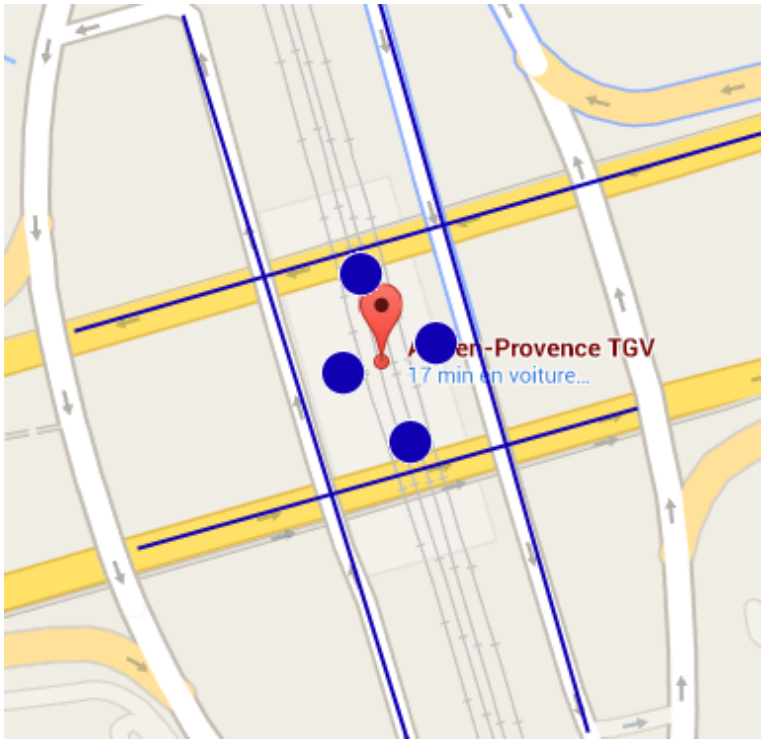


FIGURE 56 – Gare d'Aix TGV : 4 entrées (dont deux dans tunnels)

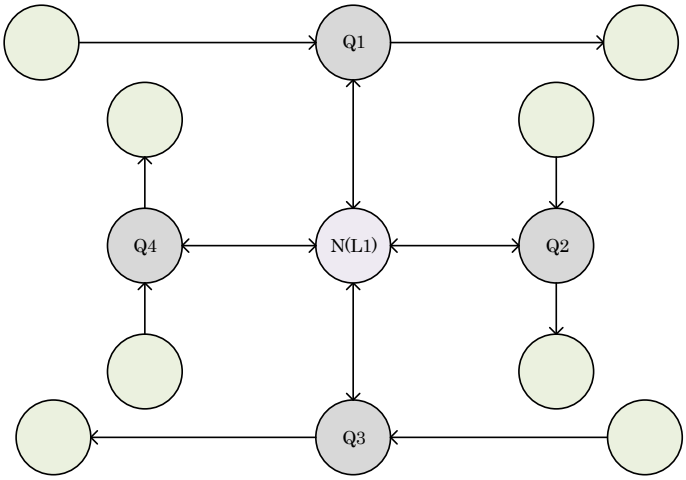


FIGURE 57 – Exemple de graphe d'un lieu snappé à 4 tronçons de voirie

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *Communiqué le jour de la soutenance*

Alexandre IGLESIAS

Multicriteria trip planning in multimodal transportation networks

Speciality: Computer Science

Keywords: Shortest paths, public transportation trip planner, multimodal, multicriteria

Abstract:

The work carried out in this industrial thesis for Cityway, a company specializing in information technologies applied to mobility, is aimed at improving its trip planner engine.
We first established a comprehensive literature review on the subject, which we used to get perspective on Cityway's models and algorithms. This enabled us to help the company take a step back from its product and justify the research axes chosen for our work.
We then looked at the multicriteria aspect of the problem. Indeed, the trip planner is based on the Dijkstra algorithm, making it possible to find paths that minimize a weighted sum of criteria. We have developed a multilabel algorithm to preserve and extend several labels on the same node. Despite a small increase in computation times, the experiments carried out in a bicriteria application of this new algorithm have satisfactory results.
We also worked on the generation and selection of alternative trips. The developed generation processes are based on either the monolabel or the multilabel algorithm. The selection process is based on the definition of a distance between trips and clustering methods.
Finally, we studied the lexicographic criterion of minimum duration in the bicriteria case. For a journey to be interesting, it must be optimal on the usual criteria, but it also needs to be as short as possible. The demonstration of certain properties on this criterion made it possible to reduce calculation times which were initially too long.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *Communiqué le jour de la soutenance*

Alexandre Iglesias

Calcul d'itinéraire multicritère en transport multimodal

Spécialité: Informatique

Mots clefs : Plus court chemin, transport en commun, multimodal, multicritère

Résumé :

Les travaux effectués dans cette thèse industrielle concernent l'amélioration du calculateur d'itinéraire de Cityway, société spécialisée dans les technologies de l'information appliquées à la mobilité.
Nous avons d'abord établi un état de l'art exhaustif, accompagné d'une mise en perspective de l'existant Cityway avec celui-ci. Cela nous a permis d'aider l'entreprise à prendre du recul sur son produit et de justifier les axes de recherche choisis pour nos travaux.
Nous nous sommes ensuite intéressés à l'aspect multicritère du problème. En effet, le calculateur, basé sur l'algorithme de Dijkstra, permet de trouver des trajets minimisant une somme pondérée de critères. Nous avons développé un algorithme multilabel permettant de conserver et étendre plusieurs labels au même nœud. Malgré une légère augmentation des temps de calculs, des résultats satisfaisants ont été obtenus dans une application bicritère de ce nouvel algorithme.
Nous avons également travaillé sur la génération et la sélection de trajets alternatifs. La génération s'appuie sur les algorithmes monolabel ou multilabel. La sélection s'appuie quant à elle sur la définition d'une distance entre les solutions et des méthodes de regroupement.
Enfin, nous nous sommes intéressés à l'optimisation du calcul du critère lexicographique de durée minimale dans le cas bicritère. Pour qu'un trajet soit intéressant, il faut qu'il soit optimal sur les critères usuels, mais aussi qu'il dure le moins longtemps possible. L'utilisation de certaines propriétés sur ce critère permet de réduire des temps de calcul initialement trop longs.